HTML

Living Standard — Last Updated 6 October 2026

One-Page Version html.spec.whatwg.org	Multipage Version /multipage
Version for Web Devs /dev	PDF Version /print.pdf
Translations 日本語 • 简体中文	FAQ on GitHub
Chat on Matrix	Contribute on GitHub whatwg/html repository
Commits on GitHub	Snapshot as of this commit
Open Issues filed on GitHub	Open an Issue whatwg.org/newbug
Tests web-platform-tests html/	Issues for Tests ongoing work

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 13
7 Loading web pages 14
8 Web application APIs 17
9 Communication 19
10 Web workers 19
11 Worklets 20
12 Web storage 20
13 The HTML syntax 21
14 The XML syntax 24
15 Rendering 24
16 Obsolete features 25
17 IANA considerations 25
Index 25
References 25
Acknowledgments 25
Intellectual property rights 25

Full table of contents

1 Introduction 26
1.1 Where does this specification fit? 26
1.2 Is this HTML5? 26
1.3 Background 27
1.4 Audience 27
1.5 Scope 27
1.6 History 27
1.7 Design notes 28

1.7.1 Serializability of script execution 29
1.7.2 Extensibility 29
1.8 HTML vs XML syntax 29
1.9 Structure of this specification 30
1.9.1 How to read this specification 31
1.9.2 Typographic conventions 31
1.10 A quick introduction to HTML 32
1.10.1 Writing secure applications with HTML 35
1.10.2 Common pitfalls to avoid when using the scripting APIs 36
1.10.3 How to catch mistakes when writing HTML: validators and conformance checkers 37
1.11 Conformance requirements for authors 37
1.11.1 Presentational markup 37
1.11.2 Syntax errors 38
1.11.3 Restrictions on content models and on attribute values 40
1.12 Suggested reading 42

2 Common infrastructure 44
2.1 Terminology 44
2.1.1 Parallelism 44
2.1.2 Resources 46
2.1.3 XML compatibility 46
2.1.4 DOM trees 46
2.1.5 Scripting 47
2.1.6 Plugins 48
2.1.7 Character encodings 48
2.1.8 Conformance classes 48
2.1.9 Dependencies 51
2.1.10 Extensibility 77
2.1.11 Interactions with XPath and XSLT 77
2.2 Policy-controlled features 79
2.3 Common microsyntaxes 79
2.3.1 Common parser idioms 79
2.3.2 Boolean attributes 79
2.3.3 Keywords and enumerated attributes 80
2.3.4 Numbers 80
2.3.4.1 Signed integers 80
2.3.4.2 Non-negative integers 81
2.3.4.3 Floating-point numbers 81
2.3.4.4 Percentages and lengths 83
2.3.4.5 Nonzero percentages and lengths 84
2.3.4.6 Lists of floating-point numbers 84
2.3.4.7 Lists of dimensions 85
2.3.5 Dates and times 86
2.3.5.1 Months 86
2.3.5.2 Dates 87
2.3.5.3 Yearless dates 87
2.3.5.4 Times 88
2.3.5.5 Local dates and times 89
2.3.5.6 Time zones 90
2.3.5.7 Global dates and times 91
2.3.5.8 Weeks 93

2.3.5.9 Durations ... 94
2.3.5.10 Vaguer moments in time ... 97
2.3.6 Legacy colors ... 98
2.3.7 Space-separated tokens ... 99
2.3.8 Comma-separated tokens ... 99
2.3.9 References ... 99
2.3.10 Media queries ... 100
2.3.11 Unique internal values ... 100
2.4 URLs ... 100
2.4.1 Terminology ... 100
2.4.2 Parsing URLs ... 100
2.4.3 Document base URLs ... 101
2.5 Fetching resources ... 102
2.5.1 Terminology ... 102
2.5.2 Determining the type of a resource ... 102
2.5.3 Extracting character encodings from `meta` elements ... 103
2.5.4 CORS settings attributes ... 103
2.5.5 Referrer policy attributes ... 104
2.5.6 Nonce attributes ... 104
2.5.7 Lazy loading attributes ... 105
2.5.8 Blocking attributes ... 107
2.5.9 Fetch priority attributes ... 108
2.6 Common DOM interfaces ... 108
2.6.1 Reflecting content attributes in IDL attributes ... 108
2.6.2 Using reflect via IDL extended attributes ... 114
2.6.3 Using reflect in specifications ... 115
2.6.4 Collections ... 116
2.6.4.1 The `HTMLAllCollection` interface ... 116
2.6.4.1.1 [[Call]] (*thisArgument*, *argumentsList*) ... 117
2.6.4.2 The `HTMLFormControlsCollection` interface ... 117
2.6.4.3 The `HTMLOptionsCollection` interface ... 119
2.6.5 The `DOMStringList` interface ... 121
2.7 Safe passing of structured data ... 122
2.7.1 Serializable objects ... 122
2.7.2 Transferable objects ... 123
2.7.3 StructuredSerializeInternal (*value*, *forStorage* [, *memory*]) ... 124
2.7.4 StructuredSerialize (*value*) ... 128
2.7.5 StructuredSerializeForStorage (*value*) ... 128
2.7.6 StructuredDeserialize (*serialized*, *targetRealm* [, *memory*]) ... 128
2.7.7 StructuredSerializeWithTransfer (*value*, *transferList*) ... 131
2.7.8 StructuredDeserializeWithTransfer (*serializeWithTransferResult*, *targetRealm*) ... 132
2.7.9 Performing serialization and transferring from other specifications ... 133
2.7.10 Structured cloning API ... 134

3 Semantics, structure, and APIs of HTML documents ... 135
3.1 Documents ... 135
3.1.1 The `Document` object ... 135
3.1.2 The `DocumentOrShadowRoot` interface ... 137
3.1.3 Ancestor origins ... 137
3.1.4 Resource metadata management ... 139

3.1.5 Reporting document loading status ... 141
3.1.6 Render-blocking mechanism ... 142
3.1.7 DOM tree accessors ... 142
3.2 Elements ... 146
3.2.1 Semantics ... 146
3.2.2 Elements in the DOM ... 148
3.2.3 HTML element constructors ... 151
3.2.4 Element definitions ... 154
3.2.4.1 Attributes ... 154
3.2.5 Content models ... 155
3.2.5.1 The "nothing" content model ... 155
3.2.5.2 Kinds of content ... 155
3.2.5.2.1 Metadata content ... 156
3.2.5.2.2 Flow content ... 157
3.2.5.2.3 Sectioning content ... 157
3.2.5.2.4 Heading content ... 157
3.2.5.2.5 Phrasing content ... 157
3.2.5.2.6 Embedded content ... 158
3.2.5.2.7 Interactive content ... 158
3.2.5.2.8 Palpable content ... 158
3.2.5.2.9 Script-supporting elements ... 159
3.2.5.3 Transparent content models ... 159
3.2.5.4 Paragraphs ... 159
3.2.6 Global attributes ... 162
3.2.6.1 The `title` attribute ... 164
3.2.6.2 The `lang` and `xml:lang` attributes ... 165
3.2.6.3 The `translate` attribute ... 166
3.2.6.4 The `dir` attribute ... 167
3.2.6.5 The `style` attribute ... 171
3.2.6.6 Embedding custom non-visible data with the `data-*` attributes ... 171
3.2.7 The `innerText` and `outerText` properties ... 174
3.2.8 Requirements relating to the bidirectional algorithm ... 177
3.2.8.1 Authoring conformance criteria for bidirectional-algorithm formatting characters ... 177
3.2.8.2 User agent conformance criteria ... 177
3.2.9 Requirements related to ARIA and to platform accessibility APIs ... 178

4 The elements of HTML ... 179
4.1 The document element ... 179
4.1.1 The `html` element ... 179
4.2 Document metadata ... 180
4.2.1 The `head` element ... 180
4.2.2 The `title` element ... 181
4.2.3 The `base` element ... 182
4.2.4 The `link` element ... 184
4.2.4.1 Processing the `media` attribute ... 189
4.2.4.2 Processing the `type` attribute ... 189
4.2.4.3 Fetching and processing a resource from a `link` element ... 190
4.2.4.4 Processing `Link` headers ... 191
4.2.4.5 Early hints ... 194
4.2.4.6 Providing users with a means to follow hyperlinks created using the `link` element ... 196
4.2.5 The `meta` element ... 196
4.2.5.1 Standard metadata names ... 198

4.2.5.2 Other metadata names ... 201
4.2.5.3 Pragma directives ... 202
4.2.5.4 Specifying the document's character encoding ... 206
4.2.6 The style element ... 207
4.2.7 Interactions of styling and scripting ... 211
4.3 Sections ... 212
4.3.1 The body element ... 212
4.3.2 The article element ... 214
4.3.3 The section element ... 216
4.3.4 The nav element ... 219
4.3.5 The aside element ... 222
4.3.6 The h1, h2, h3, h4, h5, and h6 elements ... 224
4.3.7 The hgroup element ... 225
4.3.8 The header element ... 226
4.3.9 The footer element ... 227
4.3.10 The address element ... 230
4.3.11 Headings and outlines ... 231
4.3.11.1 Heading levels & offsets ... 232
4.3.11.2 Sample outlines ... 233
4.3.11.3 Exposing outlines to users ... 236
4.3.12 Usage summary ... 236
4.3.12.1 Article or section? ... 238
4.4 Grouping content ... 238
4.4.1 The p element ... 238
4.4.2 The hr element ... 241
4.4.3 The pre element ... 242
4.4.4 The blockquote element ... 244
4.4.5 The ol element ... 247
4.4.6 The ul element ... 249
4.4.7 The menu element ... 250
4.4.8 The li element ... 251
4.4.9 The dl element ... 253
4.4.10 The dt element ... 257
4.4.11 The dd element ... 258
4.4.12 The figure element ... 259
4.4.13 The figcaption element ... 262
4.4.14 The main element ... 262
4.4.15 The search element ... 264
4.4.16 The div element ... 266
4.5 Text-level semantics ... 267
4.5.1 The a element ... 267
4.5.2 The em element ... 270
4.5.3 The strong element ... 271
4.5.4 The small element ... 272
4.5.5 The s element ... 274
4.5.6 The cite element ... 275
4.5.7 The q element ... 277
4.5.8 The dfn element ... 278
4.5.9 The abbr element ... 279
4.5.10 The ruby element ... 281

4.5.11 The rt element ... 287
4.5.12 The rp element ... 288
4.5.13 The data element ... 289
4.5.14 The time element ... 290
4.5.15 The code element ... 297
4.5.16 The var element ... 298
4.5.17 The samp element ... 299
4.5.18 The kbd element ... 300
4.5.19 The sub and sup elements ... 301
4.5.20 The i element ... 302
4.5.21 The b element ... 303
4.5.22 The u element ... 304
4.5.23 The mark element ... 305
4.5.24 The bdi element ... 307
4.5.25 The bdo element ... 309
4.5.26 The span element ... 309
4.5.27 The br element ... 310
4.5.28 The wbr element ... 311
4.5.29 Usage summary ... 312
4.6 Links ... 313
4.6.1 Introduction ... 313
4.6.2 Links created by a and area elements ... 314
4.6.3 API for hyperlink elements ... 315
4.6.4 API for a and area elements ... 320
4.6.5 Following hyperlinks ... 321
4.6.6 Downloading resources ... 322
4.6.7 Hyperlink auditing ... 325
4.6.7.1 The `Ping-From` and `Ping-To` headers ... 326
4.6.8 Link types ... 326
4.6.8.1 Link type "alternate" ... 327
4.6.8.2 Link type "author" ... 329
4.6.8.3 Link type "bookmark" ... 329
4.6.8.4 Link type "canonical" ... 330
4.6.8.5 Link type "dns-prefetch" ... 330
4.6.8.6 Link type "expect" ... 330
4.6.8.7 Link type "external" ... 332
4.6.8.8 Link type "help" ... 332
4.6.8.9 Link type "icon" ... 332
4.6.8.10 Link type "license" ... 334
4.6.8.11 Link type "manifest" ... 334
4.6.8.12 Link type "modulepreload" ... 335
4.6.8.13 Link type "nofollow" ... 337
4.6.8.14 Link type "noopener" ... 337
4.6.8.15 Link type "noreferrer" ... 338
4.6.8.16 Link type "opener" ... 338
4.6.8.17 Link type "pingback" ... 338
4.6.8.18 Link type "preconnect" ... 339
4.6.8.19 Link type "prefetch" ... 340
4.6.8.20 Link type "preload" ... 340
4.6.8.21 Link type "privacy-policy" ... 344
4.6.8.22 Link type "search" ... 344
4.6.8.23 Link type "stylesheet" ... 344

4.6.8.24 Link type "tag" 346
4.6.8.25 Link Type "terms-of-service" 347
4.6.8.26 Sequential link types 347
4.6.8.26.1 Link type "next" 348
4.6.8.26.2 Link type "prev" 348
4.6.8.27 Other link types 348
4.7 Edits 350
4.7.1 The ins element 350
4.7.2 The del element 351
4.7.3 Attributes common to ins and del elements 352
4.7.4 Edits and paragraphs 352
4.7.5 Edits and lists 353
4.7.6 Edits and tables 354
4.8 Embedded content 355
4.8.1 The picture element 355
4.8.2 The source element 355
4.8.3 The img element 359
4.8.4 Images 368
4.8.4.1 Introduction 368
4.8.4.1.1 Adaptive images 374
4.8.4.2 Attributes common to source, img, and link elements 375
4.8.4.2.1 Srcset attributes 375
4.8.4.2.2 Sizes attributes 376
4.8.4.3 Processing model 377
4.8.4.3.1 When to obtain images 378
4.8.4.3.2 Reacting to DOM mutations 379
4.8.4.3.3 The list of available images 379
4.8.4.3.4 Decoding images 379
4.8.4.3.5 Updating the image data 380
4.8.4.3.6 Preparing an image for presentation 384
4.8.4.3.7 Selecting an image source 385
4.8.4.3.8 Creating a source set from attributes 385
4.8.4.3.9 Updating the source set 386
4.8.4.3.10 Parsing a srcset attribute 387
4.8.4.3.11 Parsing a sizes attribute 389
4.8.4.3.12 Normalizing the source densities 390
4.8.4.3.13 Reacting to environment changes 390
4.8.4.4 Requirements for providing text to act as an alternative for images 391
4.8.4.4.1 General guidelines 391
4.8.4.4.2 A link or button containing nothing but the image 392
4.8.4.4.3 A phrase or paragraph with an alternative graphical representation: charts, diagrams, graphs, maps, illustrations 393
4.8.4.4.4 A short phrase or label with an alternative graphical representation: icons, logos 393
4.8.4.4.5 Text that has been rendered to a graphic for typographical effect 395
4.8.4.4.6 A graphical representation of some of the surrounding text 396
4.8.4.4.7 Ancillary images 397
4.8.4.4.8 A purely decorative image that doesn't add any information 398
4.8.4.4.9 A group of images that form a single larger picture with no links 398
4.8.4.4.10 A group of images that form a single larger picture with links 399
4.8.4.4.11 A key part of the content 399
4.8.4.4.12 An image not intended for the user 403
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 403

4.8.4.4.14 Guidance for markup generators ... 403
4.8.4.4.15 Guidance for conformance checkers ... 404
4.8.5 The iframe element ... 404
4.8.6 The embed element ... 413
4.8.7 The object element ... 416
4.8.8 The video element ... 420
4.8.9 The audio element ... 425
4.8.10 The track element ... 427
4.8.11 Media elements ... 430
4.8.11.1 Error codes ... 432
4.8.11.2 Location of the media resource ... 433
4.8.11.3 MIME types ... 434
4.8.11.4 Network states ... 434
4.8.11.5 Loading the media resource ... 435
4.8.11.6 Offsets into the media resource ... 447
4.8.11.7 Ready states ... 450
4.8.11.8 Playing the media resource ... 452
4.8.11.9 Seeking ... 459
4.8.11.10 Media resources with multiple media tracks ... 461
4.8.11.10.1 AudioTrackList and VideoTrackList objects ... 462
4.8.11.10.2 Selecting specific audio and video tracks declaratively ... 466
4.8.11.11 Timed text tracks ... 466
4.8.11.11.1 Text track model ... 466
4.8.11.11.2 Sourcing in-band text tracks ... 469
4.8.11.11.3 Sourcing out-of-band text tracks ... 470
4.8.11.11.4 Guidelines for exposing cues in various formats as text track cues ... 473
4.8.11.11.5 Text track API ... 474
4.8.11.11.6 Event handlers for objects of the text track APIs ... 479
4.8.11.11.7 Best practices for metadata text tracks ... 479
4.8.11.12 Identifying a track kind through a URL ... 481
4.8.11.13 User interface ... 481
4.8.11.14 Time ranges ... 483
4.8.11.15 The TrackEvent interface ... 484
4.8.11.16 Events summary ... 484
4.8.11.17 Security and privacy considerations ... 486
4.8.11.18 Best practices for authors using media elements ... 487
4.8.11.19 Best practices for implementers of media elements ... 487
4.8.12 The map element ... 487
4.8.13 The area element ... 489
4.8.14 Image maps ... 491
4.8.14.1 Authoring ... 491
4.8.14.2 Processing model ... 491
4.8.15 MathML ... 493
4.8.16 SVG ... 494
4.8.17 Dimension attributes ... 495
4.9 Tabular data ... 496
4.9.1 The table element ... 496
4.9.1.1 Techniques for describing tables ... 501
4.9.1.2 Techniques for table design ... 504
4.9.2 The caption element ... 504
4.9.3 The colgroup element ... 505
4.9.4 The col element ... 506
4.9.5 The tbody element ... 507

4.9.6 The thead element ... 508
4.9.7 The tfoot element ... 509
4.9.8 The tr element ... 510
4.9.9 The td element ... 512
4.9.10 The th element ... 513
4.9.11 Attributes common to td and th elements ... 515
4.9.12 Processing model ... 516
4.9.12.1 Forming a table ... 516
4.9.12.2 Forming relationships between data cells and header cells ... 519
4.9.13 Examples ... 521
4.10 Forms ... 523
4.10.1 Introduction ... 523
4.10.1.1 Writing a form's user interface ... 524
4.10.1.2 Implementing the server-side processing for a form ... 526
4.10.1.3 Configuring a form to communicate with a server ... 527
4.10.1.4 Client-side form validation ... 528
4.10.1.5 Enabling client-side automatic filling of form controls ... 529
4.10.1.6 Improving the user experience on mobile devices ... 529
4.10.1.7 The difference between the field type, the autofill field name, and the input modality ... 530
4.10.1.8 Date, time, and number formats ... 531
4.10.2 Categories ... 532
4.10.3 The form element ... 532
4.10.4 The label element ... 536
4.10.5 The input element ... 539
4.10.5.1 States of the type attribute ... 546
4.10.5.1.1 Hidden state (type=hidden) ... 546
4.10.5.1.2 Text (type=text) state and Search state (type=search) ... 546
4.10.5.1.3 Telephone state (type=tel) ... 547
4.10.5.1.4 URL state (type=url) ... 547
4.10.5.1.5 Email state (type=email) ... 548
4.10.5.1.6 Password state (type=password) ... 550
4.10.5.1.7 Date state (type=date) ... 550
4.10.5.1.8 Month state (type=month) ... 552
4.10.5.1.9 Week state (type=week) ... 553
4.10.5.1.10 Time state (type=time) ... 554
4.10.5.1.11 Local Date and Time state (type=datetime-local) ... 554
4.10.5.1.12 Number state (type=number) ... 556
4.10.5.1.13 Range state (type=range) ... 557
4.10.5.1.14 Color state (type=color) ... 559
4.10.5.1.15 Checkbox state (type=checkbox) ... 561
4.10.5.1.16 Radio Button state (type=radio) ... 561
4.10.5.1.17 File Upload state (type=file) ... 563
4.10.5.1.18 Submit Button state (type=submit) ... 565
4.10.5.1.19 Image Button state (type=image) ... 566
4.10.5.1.20 Reset Button state (type=reset) ... 568
4.10.5.1.21 Button state (type=button) ... 569
4.10.5.2 Implementation notes regarding localization of form controls ... 569
4.10.5.3 Common input element attributes ... 569
4.10.5.3.1 The maxlength and minlength attributes ... 569
4.10.5.3.2 The size attribute ... 570
4.10.5.3.3 The readonly attribute ... 570
4.10.5.3.4 The required attribute ... 571
4.10.5.3.5 The multiple attribute ... 572
4.10.5.3.6 The pattern attribute ... 573
4.10.5.3.7 The min and max attributes ... 574
4.10.5.3.8 The step attribute ... 575
4.10.5.3.9 The list attribute ... 576

4.10.5.3.10 The placeholder attribute ... 578
4.10.5.4 Common input element APIs ... 579
4.10.5.5 Common event behaviors ... 583
4.10.6 The button element ... 585
4.10.7 The select element ... 590
4.10.8 The datalist element ... 598
4.10.9 The optgroup element ... 600
4.10.10 The option element ... 601
4.10.11 The textarea element ... 606
4.10.12 The output element ... 611
4.10.13 The progress element ... 614
4.10.14 The meter element ... 616
4.10.15 The fieldset element ... 620
4.10.16 The legend element ... 623
4.10.17 The selectedcontent element ... 624
4.10.18 Form control infrastructure ... 626
4.10.18.1 A form control's value ... 626
4.10.18.2 Mutability ... 626
4.10.18.3 Association of controls and forms ... 626
4.10.19 Attributes common to form controls ... 628
4.10.19.1 Naming form controls: the name attribute ... 628
4.10.19.2 Submitting element directionality: the dirname attribute ... 629
4.10.19.3 Limiting user input length: the maxlength attribute ... 629
4.10.19.4 Setting minimum input length requirements: the minlength attribute ... 630
4.10.19.5 Enabling and disabling form controls: the disabled attribute ... 630
4.10.19.6 Form submission attributes ... 631
4.10.19.7 Autofill ... 633
4.10.19.7.1 Autofilling form controls: the autocomplete attribute ... 633
4.10.19.7.2 Processing model ... 639
4.10.20 APIs for the text control selections ... 646
4.10.21 Constraints ... 651
4.10.21.1 Definitions ... 651
4.10.21.2 Constraint validation ... 652
4.10.21.3 The constraint validation API ... 653
4.10.21.4 Security ... 656
4.10.22 Form submission ... 657
4.10.22.1 Introduction ... 657
4.10.22.2 Implicit submission ... 657
4.10.22.3 Form submission algorithm ... 658
4.10.22.4 Constructing the entry list ... 661
4.10.22.5 Selecting a form submission encoding ... 663
4.10.22.6 Converting an entry list to a list of name-value pairs ... 663
4.10.22.7 URL-encoded form data ... 664
4.10.22.8 Multipart form data ... 664
4.10.22.9 Plain text form data ... 664
4.10.22.10 The SubmitEvent interface ... 665
4.10.22.11 The FormDataEvent interface ... 665
4.10.23 Resetting a form ... 666
4.11 Interactive elements ... 666
4.11.1 The details element ... 666
4.11.2 The summary element ... 672
4.11.3 Commands ... 672
4.11.3.1 Facets ... 672

4.11.3.2 Using the a element to define a command ... 673
4.11.3.3 Using the button element to define a command ... 673
4.11.3.4 Using the input element to define a command ... 673
4.11.3.5 Using the option element to define a command ... 674
4.11.3.6 Using the accesskey attribute on a legend element to define a command ... 674
4.11.3.7 Using the accesskey attribute to define a command on other elements ... 675
4.11.4 The dialog element ... 675
4.11.5 Dialog light dismiss ... 684
4.12 Scripting ... 685
4.12.1 The script element ... 685
4.12.1.1 Processing model ... 692
4.12.1.2 Scripting languages ... 699
4.12.1.3 Restrictions for contents of script elements ... 700
4.12.1.4 Inline documentation for external scripts ... 701
4.12.1.5 Interaction of script elements and XSLT ... 702
4.12.2 The noscript element ... 703
4.12.3 The template element ... 705
4.12.3.1 Interaction of template elements with XSLT and XPath ... 711
4.12.4 The slot element ... 712
4.12.5 The canvas element ... 713
4.12.5.1 The 2D rendering context ... 718
4.12.5.1.1 Implementation notes ... 725
4.12.5.1.2 The canvas settings ... 725
4.12.5.1.3 The canvas state ... 726
4.12.5.1.4 Line styles ... 727
4.12.5.1.5 Text styles ... 731
4.12.5.1.6 Building paths ... 739
4.12.5.1.7 Path2D objects ... 745
4.12.5.1.8 Transformations ... 746
4.12.5.1.9 Image sources for 2D rendering contexts ... 748
4.12.5.1.10 Fill and stroke styles ... 749
4.12.5.1.11 Drawing rectangles to the bitmap ... 753
4.12.5.1.12 Drawing text to the bitmap ... 754
4.12.5.1.13 Drawing paths to the canvas ... 756
4.12.5.1.14 Drawing focus rings ... 759
4.12.5.1.15 Drawing images ... 760
4.12.5.1.16 Pixel manipulation ... 762
4.12.5.1.17 Compositing ... 766
4.12.5.1.18 Image smoothing ... 767
4.12.5.1.19 Shadows ... 767
4.12.5.1.20 Filters ... 768
4.12.5.1.21 Working with externally-defined SVG filters ... 769
4.12.5.1.22 Drawing model ... 769
4.12.5.1.23 Best practices ... 770
4.12.5.1.24 Examples ... 770
4.12.5.2 The ImageBitmap rendering context ... 774
4.12.5.2.1 Introduction ... 774
4.12.5.2.2 The ImageBitmapRenderingContext interface ... 775
4.12.5.3 The OffscreenCanvas interface ... 777
4.12.5.3.1 The offscreen 2D rendering context ... 781
4.12.5.4 Color spaces and color space conversion ... 782
4.12.5.5 Serializing bitmaps to a file ... 782
4.12.5.6 Security with canvas elements ... 783
4.12.5.7 Premultiplied alpha and the 2D rendering context ... 784
4.13 Custom elements ... 785
4.13.1 Introduction ... 785
4.13.1.1 Creating an autonomous custom element ... 785

4.13.1.2 Creating a form-associated custom element ... 786
4.13.1.3 Creating a custom element with default accessible roles, states, and properties ... 787
4.13.1.4 Creating a customized built-in element ... 788
4.13.1.5 Drawbacks of autonomous custom elements ... 789
4.13.1.6 Upgrading elements after their creation ... 791
4.13.1.7 Scoped custom element registries ... 792
4.13.1.8 Exposing custom element states ... 792
4.13.2 Requirements for custom element constructors and reactions ... 794
4.13.2.1 Preserving custom element state when moved ... 795
4.13.3 Core concepts ... 796
4.13.4 The `CustomElementRegistry` interface ... 798
4.13.5 Upgrades ... 803
4.13.6 Custom element reactions ... 805
4.13.7 Element internals ... 808
4.13.7.1 The `ElementInternals` interface ... 809
4.13.7.2 Shadow root access ... 810
4.13.7.3 Form-associated custom elements ... 810
4.13.7.4 Accessibility semantics ... 812
4.13.7.5 Custom state pseudo-class ... 813
4.14 Common idioms without dedicated elements ... 814
4.14.1 Breadcrumb navigation ... 814
4.14.2 Tag clouds ... 814
4.14.3 Conversations ... 815
4.14.4 Footnotes ... 817
4.15 Disabled elements ... 819
4.16 Matching HTML elements using selectors and CSS ... 819
4.16.1 Case-sensitivity of the CSS 'attr()' function ... 819
4.16.2 Case-sensitivity of selectors ... 819
4.16.3 Pseudo-classes ... 821

5 Microdata ... 826
5.1 Introduction ... 826
5.1.1 Overview ... 826
5.1.2 The basic syntax ... 826
5.1.3 Typed items ... 829
5.1.4 Global identifiers for items ... 830
5.1.5 Selecting names when defining vocabularies ... 830
5.2 Encoding microdata ... 831
5.2.1 The microdata model ... 831
5.2.2 Items ... 831
5.2.3 Names: the `itemprop` attribute ... 833
5.2.4 Values ... 835
5.2.5 Associating names with items ... 836
5.2.6 Microdata and other namespaces ... 837
5.3 Sample microdata vocabularies ... 837
5.3.1 vCard ... 838
5.3.1.1 Conversion to vCard ... 846
5.3.1.2 Examples ... 850
5.3.2 vEvent ... 851
5.3.2.1 Conversion to iCalendar ... 856
5.3.2.2 Examples ... 857

5.3.3 Licensing works ... 858
5.3.3.1 Examples ... 859
5.4 Converting HTML to other formats ... 859
5.4.1 JSON ... 859

6 User interaction ... 862
6.1 The `hidden` attribute ... 862
6.2 Page visibility ... 864
6.2.1 The `VisibilityStateEntry` interface ... 865
6.3 Inert subtrees ... 866
6.3.1 Modal dialogs and inert subtrees ... 866
6.3.2 The `inert` attribute ... 866
6.4 Tracking user activation ... 868
6.4.1 Data model ... 868
6.4.2 Processing model ... 869
6.4.3 APIs gated by user activation ... 870
6.4.4 The `UserActivation` interface ... 870
6.4.5 User agent automation ... 871
6.5 Activation behavior of elements ... 871
6.5.1 The `ToggleEvent` interface ... 871
6.5.2 The `CommandEvent` interface ... 872
6.6 Focus ... 873
6.6.1 Introduction ... 873
6.6.2 Data model ... 874
6.6.3 The `tabindex` attribute ... 878
6.6.4 Processing model ... 880
6.6.5 Sequential focus navigation ... 884
6.6.6 Focus management APIs ... 885
6.6.7 The `autofocus` attribute ... 887
6.7 Assigning keyboard shortcuts ... 889
6.7.1 Introduction ... 889
6.7.2 The `accesskey` attribute ... 890
6.7.3 Processing model ... 891
6.8 Editing ... 892
6.8.1 Making document regions editable: The `contenteditable` content attribute ... 892
6.8.2 Making entire documents editable: the `designMode` getter and setter ... 893
6.8.3 Best practices for in-page editors ... 894
6.8.4 Editing APIs ... 894
6.8.5 Spelling and grammar checking ... 894
6.8.6 Writing suggestions ... 896
6.8.7 Autocapitalization ... 897
6.8.8 Autocorrection ... 899
6.8.9 Input modalities: the `inputmode` attribute ... 900
6.8.10 Input modalities: the `enterkeyhint` attribute ... 900
6.9 Find-in-page ... 901
6.9.1 Introduction ... 901
6.9.2 Interaction with `details` and `hidden=until-found` ... 901
6.9.3 Interaction with selection ... 902
6.10 Close requests and close watchers ... 902
6.10.1 Close requests ... 902

6.10.2 Close watcher infrastructure ... 903
6.10.3 The CloseWatcher interface ... 906
6.11 Drag and drop ... 909
6.11.1 Introduction ... 909
6.11.2 The drag data store ... 911
6.11.3 The DataTransfer interface ... 911
6.11.3.1 The DataTransferItemList interface ... 914
6.11.3.2 The DataTransferItem interface ... 916
6.11.4 The DragEvent interface ... 917
6.11.5 Processing model ... 918
6.11.6 Events summary ... 923
6.11.7 The draggable attribute ... 924
6.11.8 Security risks in the drag-and-drop model ... 924
6.12 The popover attribute ... 925
6.12.1 The popover target attributes ... 935
6.12.2 Popover light dismiss ... 937

7 Loading web pages ... 939
7.1 Supporting concepts ... 939
7.1.1 Origins ... 939
7.1.1.1 Sites ... 940
7.1.1.2 Relaxing the same-origin restriction ... 942
7.1.1.3 The Origin interface ... 943
7.1.2 Origin-keyed agent clusters ... 944
7.1.3 Cross-origin opener policies ... 945
7.1.3.1 The headers ... 946
7.1.3.2 Browsing context group switches due to opener policy ... 947
7.1.3.3 Reporting ... 950
7.1.4 Cross-origin embedder policies ... 954
7.1.4.1 The headers ... 955
7.1.4.2 Embedder policy checks ... 956
7.1.5 Sandboxing ... 957
7.1.6 iframe element referrer policy ... 960
7.1.7 Policy containers ... 960
7.2 APIs related to navigation and session history ... 961
7.2.1 Security infrastructure for Window, WindowProxy, and Location objects ... 961
7.2.1.1 Integration with IDL ... 961
7.2.1.2 Shared internal slot: [[CrossOriginPropertyDescriptorMap]] ... 962
7.2.1.3 Shared abstract operations ... 962
7.2.1.3.1 CrossOriginProperties (*O*) ... 962
7.2.1.3.2 CrossOriginPropertyFallback (*P*) ... 963
7.2.1.3.3 IsPlatformObjectSameOrigin (*O*) ... 963
7.2.1.3.4 CrossOriginGetOwnPropertyHelper (*O*, *P*) ... 963
7.2.1.3.5 CrossOriginGet (*O*, *P*, *Receiver*) ... 964
7.2.1.3.6 CrossOriginSet (*O*, *P*, *V*, *Receiver*) ... 964
7.2.1.3.7 CrossOriginOwnPropertyKeys (*O*) ... 965
7.2.2 The Window object ... 965
7.2.2.1 Opening and closing windows ... 967
7.2.2.2 Indexed access on the Window object ... 971
7.2.2.3 Named access on the Window object ... 972
7.2.2.4 Accessing related windows ... 973
7.2.2.5 Historical browser interface element APIs ... 974
7.2.2.6 Script settings for Window objects ... 976

7.2.3 The WindowProxy exotic object ... 977
7.2.3.1 [[GetPrototypeOf]] ( ) ... 977
7.2.3.2 [[SetPrototypeOf]] (*V*) ... 977
7.2.3.3 [[IsExtensible]] ( ) ... 977
7.2.3.4 [[PreventExtensions]] ( ) ... 977
7.2.3.5 [[GetOwnProperty]] (*P*) ... 977
7.2.3.6 [[DefineOwnProperty]] (*P*, *Desc*) ... 978
7.2.3.7 [[Get]] (*P*, *Receiver*) ... 978
7.2.3.8 [[Set]] (*P*, *V*, *Receiver*) ... 979
7.2.3.9 [[Delete]] (*P*) ... 979
7.2.3.10 [[OwnPropertyKeys]] ( ) ... 979
7.2.4 The Location interface ... 980
7.2.4.1 [[GetPrototypeOf]] ( ) ... 986
7.2.4.2 [[SetPrototypeOf]] (*V*) ... 986
7.2.4.3 [[IsExtensible]] ( ) ... 986
7.2.4.4 [[PreventExtensions]] ( ) ... 986
7.2.4.5 [[GetOwnProperty]] (*P*) ... 986
7.2.4.6 [[DefineOwnProperty]] (*P*, *Desc*) ... 987
7.2.4.7 [[Get]] (*P*, *Receiver*) ... 987
7.2.4.8 [[Set]] (*P*, *V*, *Receiver*) ... 987
7.2.4.9 [[Delete]] (*P*) ... 987
7.2.4.10 [[OwnPropertyKeys]] ( ) ... 987
7.2.5 The History interface ... 987
7.2.6 The navigation API ... 992
7.2.6.1 Introduction ... 992
7.2.6.2 The Navigation interface ... 995
7.2.6.3 Core infrastructure ... 996
7.2.6.4 Initializing and updating the entry list ... 997
7.2.6.5 The NavigationHistoryEntry interface ... 999
7.2.6.6 The history entry list ... 1001
7.2.6.7 Initiating navigations ... 1002
7.2.6.8 Ongoing navigation tracking ... 1006
7.2.6.9 The NavigationActivation interface ... 1012
7.2.6.10 The navigate event ... 1013
7.2.6.10.1 The NavigateEvent interface ... 1013
7.2.6.10.2 The NavigationPrecommitController interface ... 1017
7.2.6.10.3 The NavigationDestination interface ... 1018
7.2.6.10.4 Firing the event ... 1019
7.2.6.10.5 Scroll and focus behavior ... 1025
7.2.7 Event interfaces ... 1026
7.2.7.1 The NavigationCurrentEntryChangeEvent interface ... 1026
7.2.7.2 The PopStateEvent interface ... 1027
7.2.7.3 The HashChangeEvent interface ... 1027
7.2.7.4 The PageSwapEvent interface ... 1028
7.2.7.5 The PageRevealEvent interface ... 1029
7.2.7.6 The PageTransitionEvent interface ... 1029
7.2.7.7 The BeforeUnloadEvent interface ... 1030
7.2.8 The NotRestoredReasons interface ... 1030
7.3 Infrastructure for sequences of documents ... 1035
7.3.1 Navigables ... 1035
7.3.1.1 Traversable navigables ... 1037
7.3.1.2 Top-level traversables ... 1037
7.3.1.3 Child navigables ... 1038
7.3.1.4 Jake diagrams ... 1040
7.3.1.5 Related navigable collections ... 1041

7.3.1.6 Navigable destruction 1042
7.3.1.7 Navigable target names 1043
7.3.2 Browsing contexts 1046
7.3.2.1 Creating browsing contexts 1047
7.3.2.2 Related browsing contexts 1049
7.3.2.3 Groupings of browsing contexts 1050
7.3.3 Fully active documents 1051
7.4 Navigation and session history 1052
7.4.1 Session history 1053
7.4.1.1 Session history entries 1053
7.4.1.2 Document state 1054
7.4.1.3 Centralized modifications of session history 1056
7.4.1.4 Low-level operations on session history 1058
7.4.2 Navigation 1059
7.4.2.1 Supporting concepts 1060
7.4.2.2 Beginning navigation 1063
7.4.2.3 Ending navigation 1067
7.4.2.3.1 The usual cross-document navigation case 1067
7.4.2.3.2 The `javascript:` URL special case 1068
7.4.2.3.3 Fragment navigations 1071
7.4.2.3.4 Non-fetch schemes and external software 1073
7.4.2.4 Preventing navigation 1074
7.4.2.5 Aborting navigation 1076
7.4.3 Reloading and traversing 1077
7.4.4 Non-fragment synchronous "navigations" 1078
7.4.5 Populating a session history entry 1079
7.4.6 Applying the history step 1089
7.4.6.1 Updating the traversable 1090
7.4.6.2 Updating the document 1100
7.4.6.3 Revealing the document 1103
7.4.6.4 Scrolling to a fragment 1104
7.4.6.5 Persisted history entry state 1105
7.5 Document lifecycle 1106
7.5.1 Shared document creation infrastructure 1106
7.5.2 Loading HTML documents 1110
7.5.3 Loading XML documents 1110
7.5.4 Loading text documents 1111
7.5.5 Loading `multipart/x-mixed-replace` documents 1111
7.5.6 Loading media documents 1112
7.5.7 Loading a document for inline content that doesn't have a DOM 1112
7.5.8 Finishing the loading process 1114
7.5.9 Unloading documents 1114
7.5.10 Destroying documents 1116
7.5.11 Aborting a document load 1117
7.6 Speculative loading 1118
7.6.1 Speculation rules 1118
7.6.1.1 Data model 1120
7.6.1.2 Parsing 1122
7.6.1.3 Processing model 1127
7.6.2 Navigational prefetching 1132
7.6.3 The `` `Speculation-Rules` `` header 1132
7.6.4 The `` `Sec-Speculation-Tags` `` header 1133
7.6.5 Security considerations 1133

7.6.5.1 Cross-site requests 1133
7.6.5.2 Injected content 1134
7.6.5.3 IP anonymization 1134
7.6.6 Privacy considerations 1134
7.6.6.1 Heuristics and optionality 1134
7.6.6.2 State partitioning 1135
7.6.6.3 Identity joining 1135
7.7 The `X-Frame-Options` header 1136
7.8 The `Refresh` header 1137
7.9 Browser user interface considerations 1137

8 Web application APIs 1140
8.1 Scripting 1140
8.1.1 Introduction 1140
8.1.2 Agents and agent clusters 1140
8.1.2.1 Integration with the JavaScript agent formalism 1140
8.1.2.2 Integration with the JavaScript agent cluster formalism 1141
8.1.3 Realms and their counterparts 1143
8.1.3.1 Environments 1143
8.1.3.2 Environment settings objects 1144
8.1.3.3 Realms, settings objects, and global objects 1145
8.1.3.3.1 Entry 1148
8.1.3.3.2 Incumbent 1148
8.1.3.3.3 Current 1151
8.1.3.3.4 Relevant 1151
8.1.3.4 Enabling and disabling scripting 1151
8.1.3.5 Secure contexts 1152
8.1.4 Script processing model 1152
8.1.4.1 Scripts 1152
8.1.4.2 Fetching scripts 1154
8.1.4.3 Creating scripts 1161
8.1.4.4 Calling scripts 1165
8.1.4.5 Killing scripts 1166
8.1.4.6 Runtime script errors 1167
8.1.4.7 Unhandled promise rejections 1169
8.1.4.8 Import map parse results 1170
8.1.4.9 Speculation rules parse results 1170
8.1.5 Module specifier resolution 1171
8.1.5.1 The resolution algorithm 1171
8.1.5.2 Import maps 1174
8.1.5.3 Import map processing model 1177
8.1.6 JavaScript specification host hooks 1184
8.1.6.1 HostEnsureCanAddPrivateElement(*O*) 1184
8.1.6.2 HostEnsureCanCompileStrings(*realm, parameterStrings, bodyString, codeString, compilationType, parameterArgs, bodyArg*) 1184
8.1.6.3 HostGetCodeForEval(*argument*) 1185
8.1.6.4 HostPromiseRejectionTracker(*promise, operation*) 1185
8.1.6.5 HostSystemUTCEpochNanoseconds(*global*) 1185
8.1.6.6 Job-related host hooks 1185
8.1.6.6.1 HostCallJobCallback(*callback, V, argumentsList*) 1186
8.1.6.6.2 HostEnqueueFinalizationRegistryCleanupJob(*finalizationRegistry*) 1186
8.1.6.6.3 HostEnqueueGenericJob(*job, realm*) 1187
8.1.6.6.4 HostEnqueuePromiseJob(*job, realm*) 1187
8.1.6.6.5 HostEnqueueTimeoutJob(*job, realm, milliseconds*) 1188

8.1.6.6.6 HostMakeJobCallback(*callable*) 1188
8.1.6.7 Module-related host hooks 1189
8.1.6.7.1 HostGetImportMetaProperties(*moduleRecord*) 1190
8.1.6.7.2 HostGetSupportedImportAttributes() 1191
8.1.6.7.3 HostLoadImportedModule(*referrer*, *moduleRequest*, *loadState*, *payload*) 1191
8.1.7 Event loops 1193
8.1.7.1 Definitions 1193
8.1.7.2 Queuing tasks 1194
8.1.7.3 Processing model 1196
8.1.7.4 Generic task sources 1203
8.1.7.5 Dealing with the event loop from other specifications 1204
8.1.8 Events 1206
8.1.8.1 Event handlers 1206
8.1.8.2 Event handlers on elements, `Document` objects, and `Window` objects 1213
8.1.8.2.1 IDL definitions 1216
8.1.8.3 Event firing 1217
8.2 The `WindowOrWorkerGlobalScope` mixin 1218
8.3 Base64 utility methods 1219
8.4 Dynamic markup insertion 1220
8.4.1 Opening the input stream 1220
8.4.2 Closing the input stream 1222
8.4.3 `document.write()` 1222
8.4.4 `document.writeln()` 1223
8.5 DOM parsing and serialization APIs 1224
8.5.1 The `DOMParser` interface 1224
8.5.2 HTML parsing methods 1226
8.5.3 HTML serialization methods 1228
8.5.4 The `innerHTML` property 1228
8.5.5 The `outerHTML` property 1229
8.5.6 The `insertAdjacentHTML()` method 1230
8.5.7 The `createContextualFragment()` method 1231
8.5.8 The `XMLSerializer` interface 1232
8.6 HTML sanitization 1233
8.6.1 Introduction 1233
8.6.1.1 Safe and unsafe 1233
8.6.2 The `Sanitizer` interface 1233
8.6.3 Sanitizer configuration 1240
8.6.3.1 Configuration invariants 1241
8.6.4 Sanitization algorithms 1242
8.6.5 Sanitization constants 1249
8.6.6 Security considerations 1252
8.6.6.1 Server-side reflected and stored XSS 1252
8.6.6.2 DOM clobbering 1252
8.6.6.3 XSS with script gadgets 1252
8.6.6.4 Mutation XSS 1252
8.7 Timers 1253
8.8 Microtask queuing 1257
8.9 User prompts 1258
8.9.1 Simple dialogs 1258
8.9.2 Printing 1260
8.10 System state and capabilities 1261

8.10.1 The `Navigator` object ... 1261
8.10.1.1 Client identification ... 1262
8.10.1.2 Language preferences ... 1264
8.10.1.3 Browser state ... 1265
8.10.1.4 Custom scheme handlers: the `registerProtocolHandler()` method ... 1265
8.10.1.4.1 Security and privacy ... 1268
8.10.1.4.2 User agent automation ... 1268
8.10.1.5 Cookies ... 1269
8.10.1.6 PDF viewing support ... 1269
8.11 Images ... 1272
8.11.1 The `ImageData` interface ... 1272
8.11.2 The `ImageBitmap` interface ... 1275
8.12 Animation frames ... 1280

9 Communication ... 1283
9.1 The `MessageEvent` interface ... 1283
9.2 Server-sent events ... 1284
9.2.1 Introduction ... 1284
9.2.2 The `EventSource` interface ... 1285
9.2.3 Processing model ... 1287
9.2.4 The `` `Last-Event-ID` `` header ... 1288
9.2.5 Parsing an event stream ... 1288
9.2.6 Interpreting an event stream ... 1289
9.2.7 Authoring notes ... 1291
9.2.8 Connectionless push and other features ... 1292
9.2.9 Garbage collection ... 1292
9.2.10 Implementation advice ... 1293
9.3 Cross-document messaging ... 1293
9.3.1 Introduction ... 1293
9.3.2 Security ... 1294
9.3.2.1 Authors ... 1294
9.3.2.2 User agents ... 1294
9.3.3 Posting messages ... 1295
9.4 Channel messaging ... 1296
9.4.1 Introduction ... 1296
9.4.1.1 Examples ... 1296
9.4.1.2 Ports as the basis of an object-capability model on the web ... 1297
9.4.1.3 Ports as the basis of abstracting out service implementations ... 1298
9.4.2 Message channels ... 1298
9.4.3 The `MessageEventTarget` mixin ... 1299
9.4.4 Message ports ... 1299
9.4.5 Broadcasting to many ports ... 1302
9.4.6 Ports and garbage collection ... 1302
9.5 Broadcasting to other browsing contexts ... 1303

10 Web workers ... 1306
10.1 Introduction ... 1306
10.1.1 Scope ... 1306
10.1.2 Examples ... 1306
10.1.2.1 A background number-crunching worker ... 1306
10.1.2.2 Using a JavaScript module as a worker ... 1307

10.1.2.3 Shared workers introduction 1309
10.1.2.4 Shared state using a shared worker 1311
10.1.2.5 Delegation 1315
10.1.2.6 Providing libraries 1317
10.1.3 Tutorials 1320
10.1.3.1 Creating a dedicated worker 1320
10.1.3.2 Communicating with a dedicated worker 1321
10.1.3.3 Shared workers 1321
10.2 Infrastructure 1322
10.2.1 The global scope 1322
10.2.1.1 The `WorkerGlobalScope` common interface 1322
10.2.1.2 Dedicated workers and the `DedicatedWorkerGlobalScope` interface 1323
10.2.1.3 Shared workers and the `SharedWorkerGlobalScope` interface 1324
10.2.2 The event loop 1325
10.2.3 The worker's lifetime 1325
10.2.4 Processing model 1328
10.2.5 Runtime script errors 1330
10.2.6 Creating workers 1331
10.2.6.1 The `AbstractWorker` mixin 1331
10.2.6.2 Script settings for workers 1331
10.2.6.3 Dedicated workers and the `Worker` interface 1332
10.2.6.4 Shared workers and the `SharedWorker` interface 1333
10.2.7 Concurrent hardware capabilities 1336
10.3 APIs available to workers 1336
10.3.1 Importing scripts and libraries 1336
10.3.2 The `WorkerNavigator` interface 1337
10.3.3 The `WorkerLocation` interface 1337

11 Worklets 1339
11.1 Introduction 1339
11.1.1 Motivations 1339
11.1.2 Code idempotence 1339
11.1.3 Speculative evaluation 1340
11.2 Examples 1340
11.2.1 Loading scripts 1341
11.2.2 Registering a class and invoking its methods 1342
11.3 Infrastructure 1342
11.3.1 The global scope 1342
11.3.1.1 Agents and event loops 1343
11.3.1.2 Creation and termination 1343
11.3.1.3 Script settings for worklets 1344
11.3.2 The `Worklet` class 1345
11.3.3 The worklet's lifetime 1347

12 Web storage 1348
12.1 Introduction 1348
12.2 The API 1349
12.2.1 The `Storage` interface 1349
12.2.2 The `sessionStorage` getter 1351
12.2.3 The `localStorage` getter 1352
12.2.4 The `StorageEvent` interface 1352

12.3 Privacy ... 1353
12.3.1 User tracking ... 1353
12.3.2 Sensitivity of data ... 1354
12.4 Security ... 1354
12.4.1 DNS spoofing attacks ... 1354
12.4.2 Cross-directory attacks ... 1354
12.4.3 Implementation risks ... 1355

13 The HTML syntax ... 1356
13.1 Writing HTML documents ... 1356
13.1.1 The DOCTYPE ... 1356
13.1.2 Elements ... 1357
13.1.2.1 Start tags ... 1358
13.1.2.2 End tags ... 1359
13.1.2.3 Attributes ... 1359
13.1.2.4 Optional tags ... 1360
13.1.2.5 Restrictions on content models ... 1366
13.1.2.6 Restrictions on the contents of raw text and escapable raw text elements ... 1366
13.1.3 Text ... 1366
13.1.3.1 Newlines ... 1366
13.1.4 Character references ... 1366
13.1.5 CDATA sections ... 1367
13.1.6 Comments ... 1367
13.1.7 Processing instructions ... 1368
13.2 Parsing HTML documents ... 1368
13.2.1 Overview of the parsing model ... 1369
13.2.2 Parse errors ... 1370
13.2.3 The input byte stream ... 1374
13.2.3.1 Parsing with a known character encoding ... 1375
13.2.3.2 Determining the character encoding ... 1375
13.2.3.3 Character encodings ... 1381
13.2.3.4 Changing the encoding while parsing ... 1381
13.2.3.5 Preprocessing the input stream ... 1382
13.2.4 Parse state ... 1382
13.2.4.1 The insertion mode ... 1382
13.2.4.2 The stack of open elements ... 1383
13.2.4.3 The list of active formatting elements ... 1385
13.2.4.4 The element pointers ... 1386
13.2.4.5 Other parsing state flags ... 1386
13.2.5 Tokenization ... 1387
13.2.5.1 Data state ... 1388
13.2.5.2 RCDATA state ... 1388
13.2.5.3 RAWTEXT state ... 1389
13.2.5.4 Script data state ... 1389
13.2.5.5 PLAINTEXT state ... 1389
13.2.5.6 Tag open state ... 1389
13.2.5.7 End tag open state ... 1390
13.2.5.8 Tag name state ... 1390
13.2.5.9 RCDATA less-than sign state ... 1390
13.2.5.10 RCDATA end tag open state ... 1391
13.2.5.11 RCDATA end tag name state ... 1391
13.2.5.12 RAWTEXT less-than sign state ... 1391

13.2.5.13 RAWTEXT end tag open state 1392
13.2.5.14 RAWTEXT end tag name state 1392
13.2.5.15 Script data less-than sign state 1392
13.2.5.16 Script data end tag open state 1392
13.2.5.17 Script data end tag name state 1393
13.2.5.18 Script data escape start state 1393
13.2.5.19 Script data escape start dash state 1393
13.2.5.20 Script data escaped state 1394
13.2.5.21 Script data escaped dash state 1394
13.2.5.22 Script data escaped dash dash state 1394
13.2.5.23 Script data escaped less-than sign state 1395
13.2.5.24 Script data escaped end tag open state 1395
13.2.5.25 Script data escaped end tag name state 1395
13.2.5.26 Script data double escape start state 1396
13.2.5.27 Script data double escaped state 1396
13.2.5.28 Script data double escaped dash state 1396
13.2.5.29 Script data double escaped dash dash state 1397
13.2.5.30 Script data double escaped less-than sign state 1397
13.2.5.31 Script data double escape end state 1397
13.2.5.32 Before attribute name state 1398
13.2.5.33 Attribute name state 1398
13.2.5.34 After attribute name state 1399
13.2.5.35 Before attribute value state 1399
13.2.5.36 Attribute value (double-quoted) state 1399
13.2.5.37 Attribute value (single-quoted) state 1400
13.2.5.38 Attribute value (unquoted) state 1400
13.2.5.39 After attribute value (quoted) state 1401
13.2.5.40 Self-closing start tag state 1401
13.2.5.41 Bogus comment state 1401
13.2.5.42 Markup declaration open state 1402
13.2.5.43 Comment start state 1402
13.2.5.44 Comment start dash state 1402
13.2.5.45 Comment state 1402
13.2.5.46 Comment less-than sign state 1403
13.2.5.47 Comment less-than sign bang state 1403
13.2.5.48 Comment less-than sign bang dash state 1403
13.2.5.49 Comment less-than sign bang dash dash state 1403
13.2.5.50 Comment end dash state 1404
13.2.5.51 Comment end state 1404
13.2.5.52 Comment end bang state 1404
13.2.5.53 DOCTYPE state 1404
13.2.5.54 Before DOCTYPE name state 1405
13.2.5.55 DOCTYPE name state 1405
13.2.5.56 After DOCTYPE name state 1406
13.2.5.57 After DOCTYPE public keyword state 1406
13.2.5.58 Before DOCTYPE public identifier state 1407
13.2.5.59 DOCTYPE public identifier (double-quoted) state 1407
13.2.5.60 DOCTYPE public identifier (single-quoted) state 1407
13.2.5.61 After DOCTYPE public identifier state 1408
13.2.5.62 Between DOCTYPE public and system identifiers state 1408
13.2.5.63 After DOCTYPE system keyword state 1409
13.2.5.64 Before DOCTYPE system identifier state 1409
13.2.5.65 DOCTYPE system identifier (double-quoted) state 1410
13.2.5.66 DOCTYPE system identifier (single-quoted) state 1410
13.2.5.67 After DOCTYPE system identifier state 1411

13.2.5.68 Bogus DOCTYPE state 1411
13.2.5.69 CDATA section state 1411
13.2.5.70 CDATA section bracket state 1411
13.2.5.71 CDATA section end state 1412
13.2.5.72 Processing instruction open state 1412
13.2.5.73 Processing instruction target state 1412
13.2.5.74 After processing instruction target state 1413
13.2.5.75 Processing instruction data state 1413
13.2.5.76 Processing instruction questionable state 1413
13.2.5.77 Character reference state 1414
13.2.5.78 Named character reference state 1414
13.2.5.79 Ambiguous ampersand state 1414
13.2.5.80 Numeric character reference state 1415
13.2.5.81 Hexadecimal character reference start state 1415
13.2.5.82 Hexadecimal character reference state 1415
13.2.5.83 Decimal character reference state 1415
13.2.5.84 Numeric character reference end state 1416
13.2.6 Tree construction 1417
13.2.6.1 Creating and inserting nodes 1418
13.2.6.2 Parsing elements that contain only text 1423
13.2.6.3 Closing elements that have implied end tags 1423
13.2.6.4 The rules for parsing tokens in HTML content 1424
13.2.6.4.1 The "initial" insertion mode 1424
13.2.6.4.2 The "before html" insertion mode 1425
13.2.6.4.3 The "before head" insertion mode 1426
13.2.6.4.4 The "in head" insertion mode 1427
13.2.6.4.5 The "in head noscript" insertion mode 1431
13.2.6.4.6 The "after head" insertion mode 1431
13.2.6.4.7 The "in body" insertion mode 1432
13.2.6.4.8 The "text" insertion mode 1443
13.2.6.4.9 The "in table" insertion mode 1445
13.2.6.4.10 The "in table text" insertion mode 1447
13.2.6.4.11 The "in caption" insertion mode 1447
13.2.6.4.12 The "in column group" insertion mode 1448
13.2.6.4.13 The "in table body" insertion mode 1449
13.2.6.4.14 The "in row" insertion mode 1450
13.2.6.4.15 The "in cell" insertion mode 1451
13.2.6.4.16 The "in template" insertion mode 1452
13.2.6.4.17 The "after body" insertion mode 1453
13.2.6.4.18 The "in frameset" insertion mode 1453
13.2.6.4.19 The "after frameset" insertion mode 1454
13.2.6.4.20 The "after after body" insertion mode 1455
13.2.6.4.21 The "after after frameset" insertion mode 1455
13.2.6.5 The rules for parsing tokens in foreign content 1455
13.2.7 The end 1458
13.2.8 Speculative HTML parsing 1459
13.2.9 Coercing an HTML DOM into an infoset 1461
13.2.10 An introduction to error handling and strange cases in the parser 1462
13.2.10.1 Misnested tags: 1462
13.2.10.2 Misnested tags: 1463
13.2.10.3 Unexpected markup in tables 1464
13.2.10.4 Scripts that modify the page as it is being parsed 1466
13.2.10.5 The execution of scripts that are moving across multiple documents 1467
13.2.10.6 Unclosed formatting elements 1467
13.3 Serializing HTML fragments 1468
13.4 Parsing HTML fragments 1473
13.5 Named character references 1474

14 The XML syntax 1484
14.1 Writing documents in the XML syntax 1484
14.2 Parsing XML documents 1484
14.3 Serializing XML fragments 1486
14.4 Parsing XML fragments 1487

15 Rendering 1488
15.1 Introduction 1488
15.2 The CSS user agent style sheet and presentational hints 1489
15.3 Non-replaced elements 1489
15.3.1 Hidden elements 1489
15.3.2 The page 1490
15.3.3 Flow content 1491
15.3.4 Phrasing content 1493
15.3.5 Bidirectional text 1495
15.3.6 Sections and headings 1495
15.3.7 Lists 1496
15.3.8 Tables 1497
15.3.9 Margin collapsing quirks 1502
15.3.10 Form controls 1502
15.3.11 The `hr` element 1503
15.3.12 The `fieldset` and `legend` elements 1504
15.4 Replaced elements 1507
15.4.1 Embedded content 1507
15.4.2 Images 1508
15.4.3 Attributes for embedded content and images 1509
15.4.4 Image maps 1510
15.5 Widgets 1511
15.5.1 Native appearance 1511
15.5.2 Writing mode 1511
15.5.3 Button layout 1511
15.5.4 The `button` element 1511
15.5.5 The `details` and `summary` elements 1512
15.5.6 The `input` element as a text entry widget 1513
15.5.7 The `input` element as domain-specific widgets 1513
15.5.8 The `input` element as a range control 1514
15.5.9 The `input` element as a color well 1514
15.5.10 The `input` element as a checkbox and radio button widgets 1514
15.5.11 The `input` element as a file upload control 1515
15.5.12 The `input` element as a button 1515
15.5.13 The `marquee` element 1515
15.5.14 The `meter` element 1516
15.5.15 The `progress` element 1517
15.5.16 The `select` element 1517
15.5.17 The `textarea` element 1523
15.6 Frames and framesets 1523
15.7 Interactive media 1525
15.7.1 Links, forms, and navigation 1525
15.7.2 The `title` attribute 1525

15.7.3 Editing hosts 1526
15.7.4 Text rendered in native user interfaces 1526
15.8 Print media 1528
15.9 Unstyled XML documents 1528

16 Obsolete features 1529
16.1 Obsolete but conforming features 1529
16.1.1 Warnings for obsolete but conforming features 1529
16.2 Non-conforming features 1530
16.3 Requirements for implementations 1535
16.3.1 The `marquee` element 1535
16.3.2 Frames 1537
16.3.3 Other elements, attributes and APIs 1538

17 IANA considerations 1546
17.1 `text/html` 1546
17.2 `multipart/x-mixed-replace` 1547
17.3 `application/xhtml+xml` 1548
17.4 `text/ping` 1549
17.5 `application/microdata+json` 1550
17.6 `application/speculationrules+json` 1551
17.7 `text/event-stream` 1552
17.8 `web+` scheme prefix 1553

Index 1554
Elements 1554
Element content categories 1561
Attributes 1562
Element interfaces 1570
All interfaces 1572
Events 1574
HTTP headers 1576
MIME types 1577

References 1579

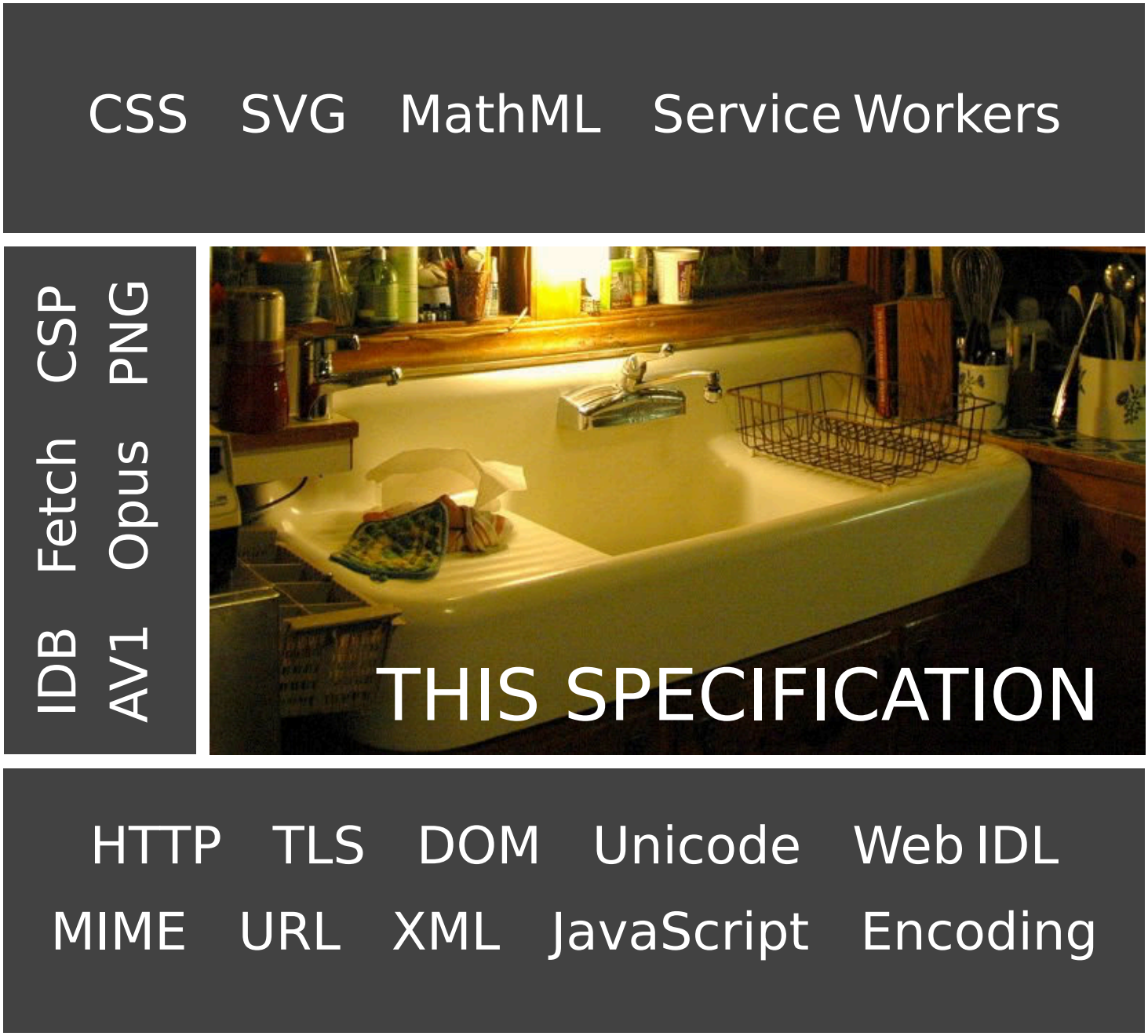
Acknowledgments 1590

Intellectual property rights 1594

1 Introduction §[p26]

1.1 Where does this specification fit? §[p26]

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:



1.2 Is this HTML5? §[p26]

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 §p27

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 §p27

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 §p27

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 §p27

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 §[p28]

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 §p29

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[p1332], the intent is that the behavior of implementations can be thought of as completely serializing the execution of all scripts in all globals.

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 Extensibility §p29

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`[p162] 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-*=""`[p171] attributes. These are guaranteed to never be touched by browsers, and allow scripts to include data on HTML elements that scripts can then look for and process.
- Authors can use the `<meta name="" content="">`[p196] mechanism to include page-wide metadata.
- Authors can use the `rel=""`[p314] mechanism to annotate links with specific meanings by registering extensions to the predefined set of link types[p348]. This is also used by microformats.
- Authors can embed raw data using the `<script type="">`[p685] mechanism with a custom type, for further handling by inline or server-side scripts.
- 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 `itemscope=""`[p831] and `itemprop=""`[p833] attributes) to embed nested name-value pairs of data to be shared with other applications and sites.
- Authors can define, share, and use custom elements[p796] to extend the vocabulary of HTML. The requirements of valid custom element names[p797] ensure forward compatibility (since no elements will be added to HTML, SVG, or MathML with hyphen-containing local names in the future).

1.8 HTML vs XML syntax §p29

This section is non-normative.

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

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

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

The first such concrete syntax is the HTML syntax. This is the format suggested for most authors. It is compatible with most legacy web browsers. If a document is transmitted with the `text/html`[p1546] 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`[p1548], 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.

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 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`[p703] 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.

1.9 Structure of this specification §[p30]

This section is non-normative.

This specification is divided into the following major sections:

Introduction[p26]

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

Common infrastructure[p44]

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

Semantics, structure, and APIs of HTML documents[p135]

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[p179]

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[p826]

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[p862]

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[p939]

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.

Web application APIs[p1140]

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

Web workers[p1306]

This section defines an API for background threads in JavaScript.

Worklets p1339

This section defines infrastructure for APIs that need to run JavaScript separately from the main JavaScript execution environment.

The communication APIs p1283

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` p1285, and a two-way full-duplex socket protocol for scripts known as Web Sockets.

Web storage p1348

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

The HTML syntax p1356

The XML syntax p1484

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 p1488

This section defines the default rendering rules for web browsers.

There are also some appendices, listing obsolete features p1529 and IANA considerations p1546, and several indices.

1.9.1 How to read this specification §p31

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 p80" 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 p80" 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 p80 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 §p31

This is a definition, requirement, or explanation.

Note

This is a note.

Example

This is an example.

This is an open issue.

⚠Warning!

This is a warning.

IDL

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

For web developers (non-normative)

***variable* = *object*.*method*[p32](*[optionalArgument]*)**

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

CSS

```
/* 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[p32] or like *this*[p32].

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`[p32].

Other code fragments are marked up `like this`.

Variables are marked up like *this*.

In an algorithm, steps in synchronous sections[p1201] 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 §[p32]

This section is non-normative.

A basic HTML document looks like this:

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

HTML documents consist of a tree of elements and text. Each element is denoted in the source by a start tag[p1358], such as "<body>", and an end tag[p1359], such as "</body>". (Certain start tags and end tags can in certain cases be omitted[p1360] and are implied by other tags.)

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

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

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

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

Elements can have attributes, which control how the elements work. In the example below, there is a hyperlink[p313], formed using the a[p267] element and its href[p314] attribute:

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

Attributes[p1359] are placed inside the start tag, and consist of a name[p1359] and a value[p1359], separated by an "=" character. The attribute value can remain unquoted[p1359] if it doesn't contain ASCII whitespace or any of " ' ` = < or >. Otherwise, it has to be quoted using either single or double quotes. The value, along with the "=" character, can be omitted altogether if the value is the empty string.

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

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

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

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

The markup snippet at the top of this section[p32] would be turned into the following DOM tree:

- DOCTYPE: html
- **html**[p179] **lang**[p165]="en"
  - **head**[p180]
    - #text: ↵␣␣
    - **title**[p181]
      - #text: *Sample page*
    - #text: ↵␣
  - #text: ↵␣
  - **body**[p212]
    - #text: ↵␣␣
    - **h1**[p224]
      - #text: *Sample page*
    - #text: ↵␣␣
    - **p**[p238]
      - #text: *This is a*
      - **a**[p267] **href**[p314]="demo.html"
        - #text: *simple*
      - #text: *sample.*
    - #text: ↵␣␣
    - #comment: *this is a comment*
    - #text: ↵␣↵

The document element of this tree is the html[p179] element, which is the element always found in that position in HTML documents. It

contains two elements, head[p180] and body[p212], 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[p180] start tag ends up being dropped silently, and all the whitespace after the body[p212] end tag ends up placed at the end of the body[p212].

The head[p180] element contains a title[p181] element, which itself contains a Text node with the text "Sample page". Similarly, the body[p212] element contains an h1[p224] element, a p[p238] 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[p685] element or using event handler content attributes[p1207]. For example, here is a form with a script that sets the value of the form's output[p611] 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[p267] element in the tree above) can have its "href[p314]" 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>
</html>
```

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

1.10.1 Writing secure applications with HTML §[p35]

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][p1584]

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:

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

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

```
https://example.com/message.cgi?say=%3Cscript%3Ealert%28%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`[p359], it is important to safelist any provided attributes as well. If one allowed all attributes then an attacker could, for instance, use the `onload`[p1215] 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:`[p1068]", but user agents can implement (and indeed, have historically implemented) others.
- Allowing a `base`[p182] element to be inserted means any `script`[p685] 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[p404] 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[p966] object to the value of the top[p973] attribute).

1.10.2 Common pitfalls to avoid when using the scripting APIs §[p36]

This section is non-normative.

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

On the other hand, parsing of HTML files happens 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[p1207], or create the element and add the event handlers in the same script. The latter is safe because, as mentioned earlier, scripts are run to completion before further events can fire.

Example

One way this could manifest itself is with img[p359] elements and the load[p1575] 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[p1215] handler on an img[p359] element to catch the load[p1575] event:

```
<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>
```

However, if the author first created the img[p359] element and then in a separate script added the event listeners, there's a chance that the load[p1575] event would be fired in between, leading it to be missed:

```
<!-- Do not use this style, it has a race condition! -->
 <img id="games" src="games.png" alt="Games">
 <!-- the 'load' event might fire here while the parser is taking a
      break, in which case you will not see it! -->
```

```
<script>
 var img = document.getElementById('games');
 img.onload = gamesLogoHasLoaded; // might never fire!
</script>
```

1.10.3 How to catch mistakes when writing HTML: validators and conformance checkers §p37

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 §p37

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 §p37

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`p171 attribute and the `style`p207 element. Use of the `style`p171 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[p207] 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[p303], i[p302], hr[p241], s[p274], small[p272], and u[p304].

1.11.2 Syntax errors §[p38]

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[p241] element that is an *earlier* sibling of the corresponding table[p496] 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[p1370].

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

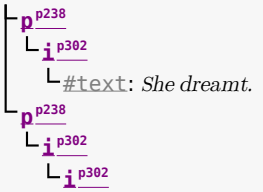
Certain syntax constructs can result in disproportionately poor performance. To discourage the use of such constructs, they are typically made non-conforming.

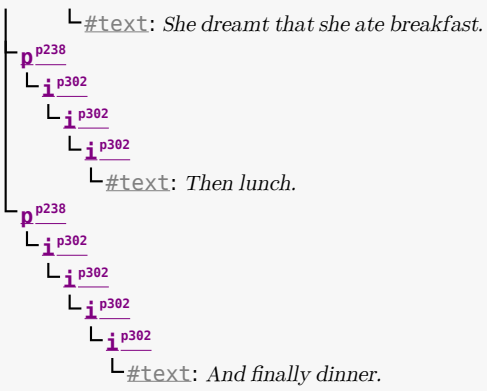
Example

For example, the following markup results in poor performance, since all the unclosed i[p302] elements have to be reconstructed in each paragraph, resulting in progressively more elements in each paragraph:

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

The resulting DOM for this fragment would be:





Errors involving fragile syntax constructs

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

Example

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

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

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

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

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

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

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

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

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

Errors involving known interoperability problems in legacy user agents

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

Example

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

Example

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

Errors that risk exposing authors to security attacks

Certain restrictions exist purely to avoid known security problems.

Example

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. [UTF7]p1587

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 h1p224 heading or an h2p224 heading:

```
<h1>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 `<capton>` instead of `<caption>`, this would be flagged as an error and the author could correct the typo immediately.

Errors that could interfere with new syntax in the future

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

Example

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 §p40

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 `section`p216 element inside a `kbd`p300 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.

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

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

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.

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

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

Example

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

Errors that indicate a likely misunderstanding of the specification

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

Example

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 circ and circle values in practice as synonyms, disallows the use of the circ value, so as to simplify tutorials and other learning aids. There would be no benefit to allowing both, but it would cause extra confusion when teaching the language.

Errors that involve peculiarities of the parser

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

Example

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

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

It is parsed exactly like the following:

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

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

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

Example

This is why, for instance, it is non-conforming to have two `id`[p162] 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`[p685] element's `src`[p686] 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`[p685] element when the `src`[p686] 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 transitionary step when migrating between the HTML and XML syntaxes.

Example

For example, there are somewhat complicated rules surrounding the `lang`[p165] 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`[p314] 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 §[p42]

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]**[p1579]

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]**[p1587]

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]**[p1587]

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]**[p1579]

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]**[p1587]

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 §p44

This specification depends on *Infra*. [INFRA]p1583

2.1 Terminology §p44

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 syntaxp1356 or the XML syntaxp1484, 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`p135 objects and their descendant DOM trees, and to serialized byte streams using the HTML syntaxp1356 or the XML syntaxp1484, 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`p135 objects can find themselves in. [DOM]p1582 (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`p1546, 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 §p44

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 loopp1193). 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 specificationsp1204.

To avoid race conditions between different in parallelp44 algorithms that operate on the same data, a parallel queuep44 can be used.

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

A parallel queuep44 has an **algorithm queue** (a queue), initially empty.

To **enqueue steps** to a parallel queuep44, enqueue the algorithm steps to the parallel queuep44's algorithm queuep44.

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

1. Let *parallelQueue* be a new parallel queuep44.
2. Run the following steps in parallelp44:

1. While true:

    1. Let *steps* be the result of dequeuing from *parallelQueue*'s algorithm queue[p44].

    2. If *steps* is not nothing, then run *steps*.

    3. Assert: running *steps* did not throw an exception, as steps running in parallel[p44] 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[p44] can themselves run other steps in in parallel[p44]. E.g., inside a parallel queue[p44] 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 created in this's relevant realm[p1151].

2. Let *global* be this's relevant global object[p1151].

3. Run the following steps in parallel[p44]:

    1. If *nameList* contains *name*, then queue a global task[p1195] on the DOM manipulation task source[p1203] given *global* to reject *p* with a `TypeError`, and abort these steps.

    2. Do some potentially lengthy work.

    3. Append *name* to *nameList*.

    4. Queue a global task[p1195] on the DOM manipulation task source[p1203] given *global* to resolve *p* with undefined.

4. Return *p*.

Two invocations of the above could run simultaneously, meaning *name* isn't in *nameList* during step 3.1, but it *might be added* before step 3.3 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[p44] and define the name-adding steps as follows:

1. Let *p* be a new promise created in this's relevant realm[p1151].

2. Let *global* be this's relevant global object[p1151].

3. Enqueue the following steps[p44] to *nameListQueue*:

    1. If *nameList* contains *name*, then queue a global task[p1195] on the DOM manipulation task source[p1203] given *global* to reject *p* with a `TypeError`, and abort these steps.

    2. Do some potentially lengthy work.

    3. Append *name* to *nameList*.

    4. Queue a global task[p1195] on the DOM manipulation task source[p1203] given *global* to resolve *p* with undefined.

4. Return *p*.

The steps would now queue and the race is avoided.

2.1.2 Resources §p46

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]p1583

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 §p46

To ease migration from HTML to XML, user agents conforming to this specification will place elements in HTML in the `http://www.w3.org/1999/xhtml` namespace, at least for the purposes of the DOM and CSS. The term "**HTML elements**" 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`p585 elements are elements with the element type `button`p585, meaning they have the local name "`button`" and (implicitly as defined above) the HTML namespace.

2.1.4 DOM trees §p46

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).

An HTML element can have specific **HTML element insertion steps**, **HTML element post-connection steps**, **HTML element removing steps**, and **HTML element moving steps** all defined for the element's local name.

The insertion steps for the HTML Standard, given *insertedNode*, are defined as the following:

1. If *insertedNode* is an element whose namespace is the HTML namespace, and this standard defines HTML element insertion steps[p46] for *insertedNode*'s local name, then run the corresponding HTML element insertion steps[p46] given *insertedNode*.

2. If *insertedNode* is a form-associated element[p532] or the ancestor of a form-associated element[p532]:

    1. If the form-associated element[p532]'s parser inserted flag[p627] is set, then return.

    2. Reset the form owner[p627] of the form-associated element[p532].

3. If *insertedNode* is an `Element` that is not on the stack of open elements[p1383] of an HTML parser[p1368], then process internal resource links[p331] given *insertedNode*'s node document.

The post-connection steps for the HTML Standard, given *insertedNode*, are defined as the following:

1. If *insertedNode* is an element whose namespace is the HTML namespace, and this standard defines HTML element post-connection steps[p46] for *insertedNode*'s local name, then run the corresponding HTML element post-connection steps[p46] given *insertedNode*.

The removing steps for the HTML Standard, given *removedNode*, *isSubtreeRoot*, and *oldAncestor*, are defined as the following:

1. Let *document* be *removedNode*'s node document.

2. If *document*'s focused area[p875] is *removedNode*, then set *document*'s focused area[p875] to *document*'s viewport, and set *document*'s relevant global object[p1151]'s navigation API[p995]'s focus changed during ongoing navigation[p1007] to false.

    Note

    This does not *perform the unfocusing steps[p882], focusing steps[p881], or focus update steps[p882], and thus no* `blur`[p1574] *or* `change`[p1575] *events are fired.*

3. If *removedNode* is an element whose namespace is the HTML namespace, and this standard defines HTML element removing steps[p46] for *removedNode*'s local name, then run the corresponding HTML element removing steps[p46] given *removedNode*, *isSubtreeRoot*, and *oldAncestor*.

4. If *removedNode* is a form-associated element[p532] with a non-null form owner[p626] and *removedNode* and its form owner[p626] are no longer in the same tree, then reset the form owner[p627] of *removedNode*.

5. If *removedNode*'s `popover`[p925] attribute is not in the No Popover[p926] state, then run the hide popover algorithm[p930] given *removedNode*, false, false, false, and null.

The moving steps for the HTML Standard, given *movedNode*, *isSubtreeRoot*, and *oldAncestor*, are defined as the following:

1. If *movedNode* is an element whose namespace is the HTML namespace, and this standard defines HTML element moving steps[p46] for *movedNode*'s local name, then run the corresponding HTML element moving steps[p46] given *movedNode*, *isSubtreeRoot*, and *oldAncestor*.

2. If *movedNode* is a form-associated element[p532] with a non-null form owner[p626] and *movedNode* and its form owner[p626] are no longer in the same tree, then reset the form owner[p627] of *movedNode*.

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[p1046] 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[p47]. 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[p47], or when its shadow-including root's browsing context[p1046] becomes null.

2.1.5 Scripting §[p47]

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 §p48

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`p135 object, but that neither act as child navigablesp1039 of the `Document`p135 nor introduce any `Node` objects to the `Document`p135'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 pluginp48.

Example

One example of a plugin would be a PDF viewer that is instantiated in a navigablep1035 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]p1584

⚠Warning!

Browsers should take extreme care when interacting with external content intended for pluginsp48. 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 pluginsp48 provides a tracking vector that increases the chances of users being uniquely identified, user agents are encouraged to support the exact same set of pluginsp48 for each user.

2.1.7 Character encodings §p48

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]p1582

2.1.8 Conformance classes §p48

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[p1484] 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`[p685] 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`[p685] element as an opaque element that forms part of the transform.

Web browsers that support the HTML syntax[p1356] 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][p1588]

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`[p685] element in the example above would still implement the `HTMLScriptElement`[p686] 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.

Note

Typical examples of non-interactive presentation user agents are printers (static UAs) and overhead displays (dynamic UAs). It is expected that most static non-interactive presentation user agents will also opt to lack scripting support[p49].

Example

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

Visual user agents that support the suggested default rendering

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

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

User agents that are designated as supporting the suggested default rendering must, while so designated, implement the rules the Rendering section[p1488] 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.

Note

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

Conformance checkers

Conformance checkers must verify that a document conforms to the applicable conformance criteria described in this specification. Automated conformance checkers are exempt from detecting errors that require interpretation of the author's intent (for example, while a document is non-conforming if the content of a `blockquote`[p244] element is not a quote, conformance checkers running without the input of human judgement do not have to check that `blockquote`[p244] elements only contain quoted material).

Conformance checkers must check that the input document conforms when parsed without a browsing context[p1046] (meaning that no scripts are run, and that the parser's scripting mode[p1387] is Disabled[p1387]), and should also check that the input document conforms when parsed with a browsing context[p1046] 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][p1579])

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

Note

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

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

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

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

Data mining tools

Applications and tools that process HTML and 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[p231] but increases the nesting level for each paragraph and does not increase the nesting level for headings[p231] would not be conforming.

Authoring tools and markup generators

Authoring tools and markup generators must generate conforming documents[p48]. 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`[p230] element for arbitrary contact information; that element can only be used for marking up contact information for its nearest `article`[p214] or `body`[p212] 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`[p230] 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`[p214] 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 roundtrip 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`[p266], `b`[p303], `i`[p302], and `span`[p309] and making liberal use of the `style`[p171] 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.

For compatibility with existing content and prior specifications, this specification describes two authoring formats: one based on XML[p1484], and one using a custom format[p1356] inspired by SGML (referred to as the HTML syntax[p1356]). Implementations must support at least one of these two formats, although supporting both is encouraged.

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

2.1.9 Dependencies §[p51]

This specification relies on several other underlying specifications.

Infra

The following terms are defined in *Infra*: [INFRA][p1583]

- The general iteration terms **while**, **continue**, and **break**.
- **Assert**
- **implementation-defined**
- **willful violation**
- **tracking vector**
- **code point** and its synonym **character**
- **surrogate**
- **scalar value**
- **tuple**
- **noncharacter**
- **string**, **code unit**, **code unit prefix**, **code unit less than**, **starts with**, **ends with**, **length**, and **code point length**
- The string equality operations **is** and **identical to**
- The **equality** of a data type, **equals**, and **itemwise equality**
- **scalar value string**
- **convert**
- **ASCII string**
- **ASCII tab or newline**
- **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**
- **isomorphic encode**

- **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 **key**, **value**, **empty**, **entry**, **exists**, **getting the value of an entry**, **setting the value of an entry**, **with default**, **removing an entry**, **clear**, **getting the keys**, **getting the values**, **sorting in descending order**, **size**, and **iterate**
- The **list** data structure and the associated definitions for **append**, **extend**, **has duplicates**, **prepend**, **replace**, **remove**, **empty**, **contains**, **size**, **indices**, **is empty**, **item**, **iterate**, **clone**, **sort in ascending order**, **sort in descending order**, **slice** (and its **from** and **to** arguments), and **reverse**
- 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**, **create**, **difference**, **intersection**, **subset** and **union**
- The **struct** specification type and the associated definition for **item**
- The **byte sequence** data structure
- The **allowed** and **blocked** singletons
- The **forgiving-base64 encode** and **forgiving-base64 decode** algorithms
- **exclusive range**
- **parse a JSON string to an Infra value**
- **HTML namespace**
- **MathML namespace**
- **SVG namespace**
- **XLink namespace**
- **XML namespace**
- **XMLNS namespace**
- **success**

Unicode and Encoding

The Unicode character set is used to represent textual data, and *Encoding* defines requirements around character encodings. [UNICODE][p1587]

Note

This specification introduces terminology[p48] based on the terms defined in those specifications, as described earlier.

The following terms are used as defined in *Encoding*: [ENCODING][p1582]

- **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 the XML syntax[p1484] 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][p1588] [XMLNS][p1588]

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.

Note

In the HTML syntax[p1356], 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][p1588]

The **Name** production is defined in *XML*. [XML][p1588]

This specification also references the **<?xml-stylesheet?>** processing instruction, defined in *Associating Style Sheets with XML*

documents. [XMLSSPI][p1589]

This specification also non-normatively mentions the `XSLTProcessor` interface and its `transformToFragment()` and `transformToDocument()` methods. [XSLTP][p1589]

URLs

The following terms are defined in *URL*: [URL][p1587]

- **host**
- **public suffix**
- **domain**
- **IP 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**
  - **blob URL entry**
- **valid URL string**
- The **cannot have a username/password/port** concept
- The **opaque path** concept
- **URL serializer** and its **exclude fragment** argument
- **URL path serializer**
- 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**
- **is special**

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

- The `about:` scheme [ABOUT][p1579]
- The `blob:` scheme [FILEAPI][p1582]
- The `data:` scheme [RFC2397][p1585]
- The `http:` scheme [HTTP][p1583]
- The `https:` scheme [HTTP][p1583]
- The `mailto:` scheme [MAILTO][p1584]
- The `sms:` scheme [SMS][p1586]
- The `urn:` scheme [URN][p1587]

Media fragment syntax is defined in *Media Fragments URI*. [MEDIAFRAG][p1584]

URL Pattern

The following terms are defined in *URL Pattern*: [URLPATTERN][p1587]

- **URL pattern**
- **match**
- **build a URL pattern from an Infra value**
- **parse a URL pattern constructor string**
- `URLPatternInit`

HTTP and related specifications

The following terms are defined in the HTTP specifications: [HTTP][p1583]

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

The following terms are defined in *HTTP State Management Mechanism*: [COOKIES][p1580]

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

The following term is defined in *Web Linking*: [WEBLINK][p1588]

- `Link` header
- **Parsing a `Link` field value**

The following terms are defined in *Structured Field Values for HTTP*: [STRUCTURED-FIELDS][p1586]

- **structured header**
- **boolean**
- **list**
- **parameters**
- **string**
- **token**

The following terms are defined in *MIME Sniffing*: [MIMESNIFF][p1584]

- **MIME type**
- **MIME type essence**
- **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**
- **image MIME type**
- **audio or video MIME type**
- **font MIME type**
- **parse a MIME type**
- **is MIME type supported by the user agent?**

Fetch

The following terms are defined in *Fetch*: [FETCH][p1582]

- **ABNF**
- `about:blank`
- `Sec-Purpose`
- An **HTTP(S) scheme**
- A URL which **is local**
- A **local scheme**
- A **fetch scheme**
- **CORS protocol**
- **environment default `User-Agent` value**
- **extract a MIME type**
- **legacy extract an encoding**
- **fetch**
- **fetch controller**
- **process the next manual redirect**
- **ok status**
- **navigation request**
- **network error**
- **aborted network error**
- `Origin` header
- `Cross-Origin-Resource-Policy` header
- **getting a structured field value**
- **header list**
- **set**
- **get, decode, and split**
- **abort**
- **cross-origin resource policy check**
- the `RequestCredentials` enumeration

- the `RequestDestination` enumeration
- the `fetch()` method
- **report timing**
- **serialize a response URL for reporting**
- **safely extracting a body**
- **incrementally reading a body**
- **processResponseConsumeBody**
- **processResponseEndOfBody**
- **processResponse**
- **useParallelQueue**
- **processEarlyHintsResponse**
- **connection pool**
- **obtain a connection**
- **determine the network partition key**
- **extract full timing info**
- **as a body**
- **response body info**
- **resolve an origin**
- **credentials**
- **reserve deferred fetch quota**
- **potentially free deferred fetch quota**
- **is offline**
- **navigation TAO check**
- **response** and its associated:
    - **type**
    - **URL**
    - **URL list**
    - **status**
    - **header list**
    - **body**
    - **body info**
    - **internal response**
    - **location URL**
    - **timing info**
    - **service worker timing info**
    - **redirect taint**
    - **timing allow passed**
    - **extract content-range values**
- **request** and its associated:
    - **URL**
    - **method**
    - **header list**
    - **body**
    - **client**
    - **URL list**
    - **current URL**
    - **reserved client**
    - **replaces client id**
    - **initiator**
    - **destination**
    - **potential destination**
    - **translating** a potential destination
    - **script-like** destinations
    - **priority**
    - **origin**
    - **referrer**
    - **mode**
    - **credentials mode**
    - **use-URL-credentials flag**
    - **unsafe-request flag**
    - **cache mode**
    - **redirect count**
    - **redirect mode**
    - **policy container**
    - **referrer policy**
    - **cryptographic nonce metadata**
    - **integrity metadata**
    - **parser metadata**
    - **reload-navigation flag**
    - **history-navigation flag**
    - **user-activation**
    - **render-blocking**
    - **initiator type**
    - **service-workers mode**
    - **traversable for user prompts**
    - **top-level navigation initiator origin**
    - **add a range header**
    - **destination type**
- **fetch timing info** and its associated:
    - **start time**
    - **end time**

The following terms are defined in *Referrer Policy*: [REFERRERPOLICY][p1585]

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

The following terms are defined in *Mixed Content*: [MIX]p1584

- ***a priori* authenticated URL**
- **mixed content**

The following terms are defined in *Subresource Integrity*: [SRI]p1586

- **parse integrity metadata**
- **the requirements of the integrity attribute**
- **get the strongest metadata from set**
- **integrity policy**
- **parse Integrity-Policy headers**

The No-Vary-Search HTTP Response Header Field

The following terms are defined in *The No-Vary-Search HTTP Response Header Field*: [NOVARYSEARCH]p1584

- `No-Vary-Search`
- **parse a URL search variance**
- **URL search variance**
- **default URL search variance**
- **equivalent modulo search variance**

Paint Timing

The following terms are defined in *Paint Timing*: [PAINTTIMING]p1584

- **mark paint timing**

Navigation Timing

The following terms are defined in *Navigation Timing*: [NAVIGATIONTIMING]p1584

- **create the navigation timing entry**
- **queue the navigation timing entry**
- `NavigationTimingType` and its "navigate", "reload", and "back_forward" values.

Resource Timing

The following terms are defined in *Resource Timing*: [RESOURCETIMING]p1585

- **Mark resource timing**

Performance Timeline

The following terms are defined in *Performance Timeline*: [PERFORMANCETIMELINE]p1585

- `PerformanceEntry` and its `name`, `entryType`, `startTime`, and `duration` attributes.
- **Queue a performance entry**

Long Animation Frames

The following terms are defined in *Long Animation Frames*: [LONGANIMATIONFRAMES]p1583

- **record task start time**
- **record task end time**
- **record rendering time**
- **record classic script creation time**
- **record classic script execution start time**
- **record module script execution start time**
- **Record pause duration**
- **record timing info for timer handler**
- **record timing info for microtask checkpoint**

Long Tasks

The following terms are defined in *Long Tasks*: [LONGTASKS]p1583

- **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]p1588

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**
- **buffer source byte length**
- **supports indexed properties**
- **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**
- **named properties object**
- **include**
- **inherit**
- **interface prototype object**
- **implements**
- **associated realm**
- **[[Realm]] field of a platform object**
- **[[GetOwnProperty]] internal method of a named properties 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 resolved with**
- **a promise rejected with**
- **wait for all**
- **upon rejection**
- **upon fulfillment**
- **mark as handled**
- `[Global]`
- `[LegacyFactoryFunction]`
- `[LegacyLenientThis]`
- `[LegacyNullToEmptyString]`
- `[LegacyOverrideBuiltIns]`
- **LegacyPlatformObjectGetOwnProperty**
- `[LegacyTreatNonObjectAsNull]`
- `[LegacyUnenumerableNamedProperties]`
- `[LegacyUnforgeable]`
- **set entries**

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

- `ArrayBuffer`
- `ArrayBufferView`
- `boolean`
- `DOMString`
- `double`
- **enumeration**
- `Float16Array`
- `Function`
- `long`
- `object`
- `Promise`
- `Uint8ClampedArray`
- `unrestricted double`
- `unsigned long`
- `USVString`
- `VoidFunction`
- `QuotaExceededError`

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"`
- `"NoModificationAllowedError"`
- `"NotFoundError"`
- `"NotSupportedError"`
- `"InvalidStateError"`
- `"SyntaxError"`
- `"InvalidAccessError"`
- `"SecurityError"`
- `"NetworkError"`
- `"AbortError"`
- `"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 `Date` object that represents a time value NaN (indicating that the object does not represent a specific instant of time).

JavaScript

Some parts of the language described by this specification only support JavaScript as the underlying scripting language. [JAVASCRIPT]p1583

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.

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**
- **clamping** a mathematical value
- **early error**
- **forward progress**
- **invariants of the essential internal methods**
- **JavaScript execution context**
- **JavaScript execution context stack**
- **realm**
- **JobCallback Record**
- **NewTarget**
- **running JavaScript execution context**
- **surrounding agent**
- **abstract closure**
- **immutable prototype exotic object**
- **Well-Known Symbols**, including **%Symbol.hasInstance%**, **%Symbol.isConcatSpreadable%**, **%Symbol.toPrimitive%**, and **%Symbol.toStringTag%**
- **Well-Known Intrinsic Objects**, including **%Array.prototype%**, **%Error.prototype%**, **%EvalError.prototype%**, **%Function.prototype%**, **%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 **BigInt**, **Boolean**, **Number**, **String**, **Symbol**, and **Object** ECMAScript language types
- The **Completion Record** specification type
- The **List** and **Record** specification types
- The **Property Descriptor** specification type
- The **ModuleRequest Record** specification type
- The **Script Record** specification type
- The **Synthetic Module Record** specification type
- The **Cyclic Module Record** specification type
- The **Source Text Module Record** specification type and its **Evaluate**, **Link** and **LoadRequestedModules** methods
- The **ArrayCreate** abstract operation
- The **Call** abstract operation
- The **ClearKeptObjects** abstract operation
- The **CleanupFinalizationRegistry** abstract operation
- The **Construct** abstract operation

- The **CopyDataBlockBytes** abstract operation
- The **CreateBuiltinFunction** abstract operation
- The **CreateByteDataBlock** abstract operation
- The **CreateDataProperty** abstract operation
- The **CreateDefaultExportSyntheticModule** abstract operation
- The **DefinePropertyOrThrow** abstract operation
- The **DetachArrayBuffer** abstract operation
- The **EnumerableOwnProperties** abstract operation
- The **FinishLoadingImportedModule** 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 **HostEnqueueFinalizationRegistryCleanupJob** abstract operation
- The **HostEnqueueGenericJob** abstract operation
- The **HostEnqueuePromiseJob** abstract operation
- The **HostEnqueueTimeoutJob** abstract operation
- The **HostEnsureCanAddPrivateElement** abstract operation
- The **HostGetSupportedImportAttributes** abstract operation
- The **HostLoadImportedModule** abstract operation
- The **HostMakeJobCallback** abstract operation
- The **HostPromiseRejectionTracker** abstract operation
- The **InitializeHostDefinedRealm** abstract operation
- The **IsArrayBufferViewOutOfBounds** 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 **OrdinaryGetOwnProperty** abstract operation
- The **OrdinaryDefineOwnProperty** abstract operation
- The **OrdinaryGet** abstract operation
- The **OrdinarySet** abstract operation
- The **OrdinaryDelete** abstract operation
- The **OrdinaryOwnPropertyKeys** abstract operation
- The **OrdinaryObjectCreate** abstract operation
- The **ParseJSONModule** 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 **SetSyntheticModuleExport** abstract operation
- The **SetImmutablePrototype** abstract operation
- The **ThrowCompletion** abstract operation
- The **ToBoolean** abstract operation
- The **ToString** abstract operation
- The **ToUint32** abstract operation
- The **TypedArrayCreate** abstract operation
- The **IsLooselyEqual** abstract operation
- The **IsStrictlyEqual** abstract operation
- The `Atomics` object
- The `Atomics.waitAsync` object
- The `Date` class
- The `FinalizationRegistry` class
- The `RegExp` class
- The `SharedArrayBuffer` class
- The `SyntaxError` class
- The `TypeError` class
- The `RangeError` class
- The `WeakRef` class
- The `eval()` function
- The `WeakRef.prototype.deref()` function
- The **[[IsHTMLDDA]]** internal slot
- `import()`
- `import.meta`
- The **HostGetImportMetaProperties** abstract operation
- The `typeof` operator
- The `delete` operator
- **The *TypedArray* Constructors** table

User agents that support JavaScript must also implement the *Dynamic Code Brand Checks* proposal. The following terms are defined there, and used in this specification: [JSDYNAMICCODEBRANDCHECKS][p1583]

- The **HostEnsureCanCompileStrings** abstract operation
- The **HostGetCodeForEval** abstract operation

User agents that support JavaScript must also implement the *Import Text* proposal. The following term is defined there, and used in this specification: [JSIMPORTTEXT][p1583]

- The **CreateTextModule** abstract operation

User agents that support JavaScript must also implement *ECMAScript Internationalization API*. [JSINTL][p1583]

User agents that support JavaScript must also implement the *Temporal* proposal. The following terms are defined there, and used in this specification: [JSTEMPORAL][p1583]

- The **HostSystemUTCEpochNanoseconds** abstract operation
- The **nsMaxInstant** and **nsMinInstant** values

WebAssembly

The following term is defined in *WebAssembly JavaScript Interface*: [WASMJS][p1587]

- `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][p1582]

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][p1582] [UIEVENTS][p1587]

In particular, the following features are defined in *DOM*: [DOM][p1582]

- `Attr` interface
- `CDATASection` interface
- `CharacterData` interface
- `Comment` interface
- `DOMImplementation` interface
- `Document` interface and its `doctype` attribute
- `DocumentOrShadowRoot` interface
- `DocumentFragment` interface
- `DocumentType` interface
- `ChildNode` interface
- `Element` interface
- `attachShadow()` method.
- An element's **shadow root**
- A shadow root's **mode**
- A shadow root's **declarative** member
- The **attach a shadow root** algorithm
- The **retargeting algorithm**
- `Node` interface
- `NodeList` interface
- `ProcessingInstruction` interface, its **target** concept, and the **get a processing instruction attribute** algorithm
- `ShadowRoot` interface
- `Text` interface
- `Range` interface
- **node document** concept
- **document type** concept
- **host** concept
- The **shadow root** concept, and its **delegates focus**, **available to element internals**, **serializable**, **clonable**, **custom element registry**, and **keep custom element registry null**.
- 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 and `supports` operation
- `createDocument()` method
- `createHTMLDocument()` method
- `createElement()` method
- `createElementNS()` method
- `getElementById()` method
- `getElementsByClassName()` method
- `append()` method
- `appendChild()` method
- `cloneNode()` method
- `moveBefore()` method
- `importNode()` method
- `preventDefault()` method

- `id` attribute
- `setAttribute()` method
- `textContent` attribute
- `children` attribute
- The **tree**, **shadow tree**, and **node tree** concepts
- The **tree order** and **shadow-including tree order** concepts
- The **element** concept
- The **child** concept
- The **root** and **shadow-including root** concepts
- The **inclusive ancestor**, **ancestor**, **descendant**, **shadow-including ancestor**, **shadow-including descendant**, **shadow-including inclusive descendant**, **shadow-including inclusive ancestor**, and **host-including inclusive ancestor** concepts
- The **first child**, **last child**, **next sibling**, **previous sibling**, and **parent** concepts
- The **parent element** concept
- 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 **slot assignment** concept
- The **slottable** concept
- The **assign slottables for a tree** algorithm
- The `slotchange` event
- The **inclusive descendant** concept
- The **find flattened slottables** algorithm
- The **manual slot assignment** concept
- The **assign a slot** algorithm
- The **pre-insert**, **ensure pre-insert validity**, **insert**, **append**, **replace**, **replace all**, **string replace all**, **remove**, and **adopt** algorithms for nodes
- The **insertion steps**,
- The **post-connection steps**, **removing steps**, **moving steps**, **adopting steps**, and **children changed steps** hooks for elements
- The **change**, **append**, **remove**, **replace**, **get an attribute by namespace and local name**, **set value**, and **remove an attribute by namespace and local name** algorithms for attributes
- The **attribute change steps** hook for attributes
- The **value** concept for attributes
- The **local name** concept for attributes
- The **namespace** concept for attributes
- The **attribute list** concept
- The **data** of a `CharacterData` node and its **replace data** algorithm
- The **child text content** of a node
- The **descendant text content** of a node
- The **name**, **public ID**, and **system ID** of a doctype
- `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 flag**
- The **dispatch** algorithm
- `EventInit` dictionary type
- `type` attribute
- An event's **target**
- `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**, **custom element registry**, **allow declarative shadow roots**, and **content type** of a `Document`[p135]
- The distinction between **XML documents** and **HTML documents**
- The terms **quirks mode**, **limited-quirks mode**, and **no-quirks mode**
- The algorithm **clone a node** with its arguments ***document***, ***subtree***, ***parent***, and ***fallbackRegistry***, and the concept of **cloning steps**
- The concept of an element's **unique identifier (ID)**
- The concept of an element's **classes**
- The term **supported tokens**
- The concept of a DOM **range**, and the terms **start node**, **start**, **end**, and **boundary point** as applied to ranges.
- The **create an element** algorithm
- The **create a document**, **create a text node**, **create a CDATA section node**, **create a doctype**, **create a comment node**, **create a processing instruction node**, and **create a document fragment** algorithms
- 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 registry**, **custom element definition**, and **is value**
- `MutationObserver` interface and **mutation observers** in general
- `AbortController` and its **signal**
- `AbortSignal`
- **aborted**
- **signal abort**
- **add**
- **valid attribute local name**
- **valid element local name**
- **is a global custom element registry**

The following features are defined in *UI Events*: [UIEVENTS][p1587]

- The `FocusEvent` interface
- The `FocusEvent` interface's `relatedTarget` attribute
- The `UIEvent` interface
- The `UIEvent` interface's `view` attribute
- `beforeinput` event
- `contextmenu` event
- `input` event
- `keydown` event
- `keypress` event
- `keyup` event

The following features are defined in *Touch Events*: [TOUCH][p1587]

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

The following features are defined in *Pointer Events*: [POINTEREVENTS][p1585]

- The `MouseEvent` interface
- The `MouseEvent` interface's `relatedTarget` attribute
- The `MouseEvent` interface's `button` attribute
- `MouseEventInit` dictionary type
- The `PointerEvent` interface
- The `PointerEvent` interface's `pointerType` attribute
- **fire a pointer event**
- `auxclick` event
- `click` event
- `dblclick` event
- `mousedown` event
- `mouseenter` event
- `mouseleave` event
- `mousemove` event
- `mouseout` event
- `mouseover` event
- `mouseup` event
- `pointerdown` event
- `pointerup` event
- `pointercancel` event
- `wheel` event

The following events are defined in *Clipboard API and events*: [CLIPBOARD-APIS][p1579]

- `copy` event
- `cut` event
- `paste` 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][p1582]

- `XML serialization`

The following features are defined in *Selection API*: [SELECTION][p1586]

- **selection**
- `Selection`

Note

User agents are encouraged to implement the features described in execCommand. *[EXECCOMMAND][p1582]*

The following features are defined in *Fullscreen API*: [FULLSCREEN][p1582]

- `requestFullscreen()`
- `fullscreenchange`
- **run the fullscreen steps**
- **fully exit fullscreen**
- **fullscreen element**
- **fullscreen flag**

High Resolution Time provides the following features: [HRT][p1582]

- **current high resolution time**
- **relative high resolution time**
- **unsafe shared current time**
- **shared monotonic clock**
- **unsafe moment**
- **duration from**
- **coarsen time**
- **current wall time**
- **Unix epoch**
- `DOMHighResTimeStamp`

File API

This specification uses the following features defined in *File API*: [FILEAPI][p1582]

- 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 **environment**
- The **obtain a blob object** algorithm

Indexed Database API

The following terms are defined in *Indexed Database API*: [INDEXEDDB][p1583]

- **cleanup Indexed Database transactions**
- `IDBVersionChangeEvent`

Media Source Extensions

The following terms are defined in *Media Source Extensions*: [MEDIASOURCE][p1584]

- `MediaSource` interface
- **detaching from a media element**

Media Capture and Streams

The following terms are defined in *Media Capture and Streams*: [MEDIASTREAM][p1584]

- `MediaStream` interface
- `MediaStreamTrack`
- **live state**
- `getUserMedia()`

Reporting

The following terms are defined in *Reporting*: [REPORTING][p1584]

- **endpoint** and its **name**
- **Queue a report**
- **report type**
- **visible to `ReportingObserver`s**

XMLHttpRequest

The following features and terms are defined in *XMLHttpRequest*: [XHR][p1588]

- The `XMLHttpRequest` interface, and its `responseXML` attribute
- The `ProgressEvent` interface, and its `lengthComputable`, `loaded`, and `total` attributes
- The `FormData` interface, and its associated **entry list**

Battery Status

The following features are defined in *Battery Status API*: [BATTERY][p1579]

- `getBattery()` method

Media Queries

Implementations must support *Media Queries*. The **<media-condition>** feature is defined therein. [MQ][p1584]

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][p1581]

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.

The following terms and features are defined in *Cascading Style Sheets* (*CSS*): [CSS][p1580]

- **viewport**
- **line box**
- **out-of-flow**
- **in-flow**
- **collapsing margins**
- **containing block**
- **inline box**
- **block box**
- The **'top'**, **'bottom'**, **'left'**, and **'right'** properties
- The **'float'** property
- The **'clear'** property
- The **'width'** property
- The **'height'** property
- The **'min-width'** property
- The **'min-height'** property
- The **'max-width'** property
- The **'max-height'** property
- The **'line-height'** property
- The **'vertical-align'** property
- The **'content'** property
- The **'inline-block'** value of the 'display' property
- The **'visibility'** property

The basic version of the **'display'** property is defined in *CSS*, and the property is extended by other CSS modules. [CSS][p1580] [CSSRUBY][p1581] [CSSTABLE][p1581]

The following terms and features are defined in *CSS Box Model*: [CSSBOX][p1580]

- **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][p1581]

- The **'margin-block'**, **'margin-block-start'**, **'margin-block-end'**, **'margin-inline'**, **'margin-inline-start'**, and **'margin-inline-end'** properties
- The **'padding-block'**, **'padding-block-start'**, **'padding-block-end'**, **'padding-inline'**, **'padding-inline-start'**, and **'padding-inline-end'** properties
- The **'border-block-width'**, **'border-block-start-width'**, **'border-block-end-width'**, **'border-inline-width'**, **'border-inline-start-width'**, **'border-inline-end-width'**, **'border-block-style'**, **'border-block-start-style'**, **'border-block-end-style'**, **'border-inline-style'**, **'border-inline-start-style'**, **'border-inline-end-style'**, **'border-block-start-color'**, **'border-block-end-color'**, **'border-inline-start-color'**, **'border-inline-end-color'**, **'border-start-start-radius'**, **'border-start-end-radius'**, **'border-end-start-radius'**, and **'border-end-end-radius'** properties

- The **'block-size'** property
- The **'inline-size'** property
- The **'inset-block-start'** property
- The **'inset-block-end'** property

The following terms and features are defined in *CSS Color*: [CSSCOLOR][p1580]

- **named color**
- **<color>**
- The **'color'** property
- The **'currentcolor'** value
- **opaque black**
- **transparent black**
- **'srgb'** color space
- **'srgb-linear'** color space
- **'display-p3'** color space
- **'display-p3-linear'** color space
- **'relative-colorimetric'** rendering intent
- **parse a CSS <color> value**
- **serialize a CSS <color> value** including **HTML-compatible serialization is requested**
- **Converting Colors**
- **'color()'**

The following terms are defined in *CSS Images*: [CSSIMAGES][p1581]

- **default object size**
- **default sizing algorithm**
- **concrete object size**
- **natural dimensions**
- **natural height**
- **natural width**
- The **'image-orientation'** property
- **'conic-gradient'**
- 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][p1581]

The following features are defined in *CSS Backgrounds and Borders*: [CSSBG][p1580]

- The **'background-color'**, **'background-image'**, **'background-repeat'**, **'background-attachment'**, **'background-position'**, **'background-clip'**, **'background-origin'**, and **'background-size'** properties
- The **'border-radius'**, **'border-top-left-radius'**, **'border-top-right-radius'**, **'border-bottom-right-radius'**, **'border-bottom-left-radius'** properties
- The **'border-image-source'**, **'border-image-slice'**, **'border-image-width'**, **'border-image-outset'**, and **'border-image-repeat'** properties

CSS Backgrounds and Borders also defines the following border properties: [CSSBG][p1580]

Border properties

	Top	Bottom	Left	Right
Width	**'border-top-width'**	**'border-bottom-width'**	**'border-left-width'**	**'border-right-width'**
Style	**'border-top-style'**	**'border-bottom-style'**	**'border-left-style'**	**'border-right-style'**
Color	**'border-top-color'**	**'border-bottom-color'**	**'border-left-color'**	**'border-right-color'**

The following features are defined in *CSS Box Alignment*: [CSSALIGN][p1580]

- 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][p1580]

- **outer display type**
- **inner 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][p1580]

- The **'flex-direction'** property
- The **'flex-wrap'** property

The following terms and features are defined in *CSS Fonts*: [CSSFONTS]p1580

- **first available font**
- The **'font-family'** property
- The **'font-weight'** property
- The **'font-size'** property
- The **'font'** property
- The **'font-kerning'** property
- The **'font-stretch'** property
- The **'font-variant-caps'** property
- The **'small-caps'** value
- The **'all-small-caps'** value
- The **'petite-caps'** value
- The **'all-petite-caps'** value
- The **'unicase'** value
- The **'titling-caps'** value
- The **'ultra-condensed'** value
- The **'extra-condensed'** value
- The **'condensed'** value
- The **'semi-condensed'** value
- The **'semi-expanded'** value
- The **'expanded'** value
- The **'extra-expanded'** value
- The **'ultra-expanded'** value

The following features are defined in *CSS Forms*: [CSSFORMS]p1580

- **'::picker'**
- **'::checkmark'**
- **'::picker-icon'**
- **picker form control identifier**
- The concept **element with default preferred size**

The following features are defined in *CSS Gaps*: [CSSGAPS]p1580

- The **'gap'** property
- The **'rule'** property
- The **'rule-break'** property
- The **'rule-inset'** property
- The **'rule-overlap'** property
- The **'rule-visibility-items'** property

The following features are defined in *CSS Grid Layout*: [CSSGRID]p1581

- The **'grid-auto-columns'** property
- The **'grid-auto-flow'** property
- The **'grid-auto-rows'** 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]p1581

- **alphabetic baseline**
- **ascent metric**
- **descent metric**
- **em-over baseline**
- **em-under baseline**
- **hanging baseline**
- **ideographic-under baseline**

The following terms and features are defined in *CSS Box Sizing*: [CSSSIZING]p1581

- **fit-content inline size**
- **'aspect-ratio'** property
- **intrinsic size**

The following features are defined in *CSS Lists and Counters*. [CSSLISTS]p1581

- **list item**
- The **'counter-reset'** property
- The **'counter-set'** property
- The **'list-style-type'** property

The following features are defined in *CSS Overflow*. [CSSOVERFLOW]p1581

- The **'overflow'** property and its **'hidden'** value

- The **'text-overflow'** property
- The term **scroll container**

The following terms and features are defined in *CSS Positioned Layout*: [CSSPOSITION]p1581

- **absolutely-positioned**
- The **'position'** property and its **'static'** value
- The **top layer** (an ordered set)
- **add an element to the top layer**
- **request an element to be removed from the top layer**
- **remove an element from the top layer immediately**
- **process top layer removals**

The following features are defined in *CSS Multi-column Layout*. [CSSMULTICOL]p1581

- The **'column-count'** property
- The **'column-fill'** property
- The **'column-width'** property

The **'ruby-base'** value of the 'display' property is defined in *CSS Ruby Layout*. [CSSRUBY]p1581

The following features are defined in *CSS Table*: [CSSTABLE]p1581

- 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]p1581

- The **content language** concept
- The **'text-transform'** property
- The **'white-space'** property
- The **'text-align'** property
- The **'letter-spacing'** property
- The **'word-spacing'** property

The following features are defined in *CSS Writing Modes*: [CSSWM]p1582

- The **'direction'** property
- The **'unicode-bidi'** property
- The **'writing-mode'** property
- The **block flow direction**, **block axis**, **inline axis**, **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]p1582

- The **'outline'** property
- The **'cursor'** property
- The **'appearance'** property, its **<compat-auto>** non-terminal value type, its **'textfield'** value, and its **'menulist-button'** value.
- The **'field-sizing'** property, and its **'content'** value.
- The concept **widget**
- The concept **native appearance**
- The concept **primitive appearance**
- The concept **base appearance**
- The **non-devolvable widget** and **devolvable widget** classification, and the related **devolved widget** state.
- The **'pointer-events'** property
- The **'user-select'** property

The algorithm to **update animations and send events** is defined in *Web Animations*. [WEBANIMATIONS]p1587

Implementations that support scripting must support the CSS Object Model. The following features and terms are defined in the CSSOM specifications: [CSSOM]p1581 [CSSOMVIEW]p1581

- **`Screen`** interface
- **`LinkStyle`** interface
- **`CSSStyleDeclaration`** interface
- **`style`** IDL attribute
- **`cssText`** attribute of `CSSStyleDeclaration`
- **`StyleSheet`** interface
- **`CSSStyleSheet`** interface
- **create a CSS style sheet**
- **remove a CSS style sheet**
- **associated CSS style sheet**
- **create a constructed `CSSStyleSheet`**
- **synchronously replace the rules of a `CSSStyleSheet`**
- **disable a 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 a target into view**
- **Scroll to the beginning of the document**
- The `resize` event
- The `scroll` event
- The `scrollend` event
- **set up browsing context features**
- The **clientX** and **clientY** extension attributes of the MouseEvent interface

The following features and terms are defined in *CSS Syntax*: [CSSSYNTAX][p1581]

- **conformant style sheet**
- **parse a list of component values**
- **parse a comma-separated list of component values**
- **component value**
- **environment encoding**
- **<whitespace-token>**

The following terms are defined in *Selectors*: [SELECTORS][p1586]

- `<selector-list>`
- **parse a selector**
- **selector**
- **type selector**
- **attribute selector**
- **pseudo-class**
- `:focus-visible` pseudo-class
- **indicate focus**
- **pseudo-element**
- **match a selector against an element**
- **scoping root**

The following features are defined in *CSS Values and Units*: [CSSVALUES][p1581]

- **<length>**
- The **'em'** unit
- The **'ex'** unit
- The **'vw'** unit
- The **'in'** unit
- The **'px'** unit
- The **'pt'** unit
- The **'attr()'** function
- The **math functions**

The following features are defined in *CSS View Transitions*: [CSSVIEWTRANSITIONS][p1582]

- **perform pending transition operations**
- **rendering suppression for view transitions**
- **activate view transition**
- `ViewTransition`
- **view transition page visibility change steps**
- **resolving inbound cross-document view-transition**
- **setting up a cross-document view-transition**
- **can navigation trigger a cross-document view-transition?**

The term **style attribute** is defined in *CSS Style Attributes*. [CSSATTR][p1580]

The following terms are defined in the *CSS Cascading and Inheritance*: [CSSCASCADE][p1580]

- **cascaded value**
- **specified value**
- **computed value**
- **used value**
- **cascade origin**

- **Author Origin**
- **User Origin**
- **User Agent Origin**
- **Animation Origin**
- **Transition Origin**
- **initial value**

The CanvasRenderingContext2D[p719] 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][p1580] [CSSFONTLOAD][p1580]

The following interfaces and terms are defined in *Geometry Interfaces*: [GEOMETRY][p1582]

- **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 **DOMPointInit** dictionary, and associated **x** and **y** members
- **Matrix multiplication**

The following terms are defined in the *CSS Scoping*: [CSSSCOPING][p1581]

- **flat tree**

The following terms and features are defined in *CSS Color Adjustment*: [CSSCOLORADJUST][p1580]

- **'color-scheme'**
- **page's supported color-schemes**

The following terms are defined in *CSS Pseudo-Elements*: [CSSPSEUDO][p1581]

- **'::details-content'**
- **'::file-selector-button'**

The following terms are defined in *CSS Containment*: [CSSCONTAIN][p1580]

- **skips its contents**
- **relevant to the user**
- **proximity to the viewport**
- **layout containment**
- **'content-visibility'** property
- **'auto'** value for 'content-visibility'

The following terms are defined in *CSS Anchor Positioning*: [CSSANCHOR][p1580]

- **implicit anchor element**

Intersection Observer

The following term is defined in *Intersection Observer*: [INTERSECTIONOBSERVER][p1583]

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

Resize Observer

The following terms are defined in *Resize Observer*: [RESIZEOBSERVER][p1583]

- **gather active resize observations at depth**
- **has active resize observations**
- **has skipped resize observations**
- **broadcast active resize observations**
- **deliver resize loop error**

WebGL

The following interfaces are defined in the WebGL specifications: [WEBGL][p1588]

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

WebGPU

The following interfaces are defined in *WebGPU*: [WEBGPU][p1588]

- `GPUCanvasContext` interface

WebVTT

Implementations may support WebVTT as a text track format for subtitles, captions, metadata, etc., for media resources. [WEBVTT]p1588

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

- **WebVTT file**
- **WebVTT file using caption or subtitle cue text**
- **WebVTT file using only nested cues**
- **WebVTT parser**
- The **rules for updating the display of WebVTT text tracks**
- The WebVTT **text track cue writing direction**
- `VTTCue` interface

ARIA

The `role` attribute is defined in *Accessible Rich Internet Applications* (*ARIA*), as are the following roles: [ARIA]p1579

- `button`
- `presentation`

In addition, the following `aria-*` content attributes are defined in *ARIA*: [ARIA]p1579

- `aria-checked`
- `aria-describedby`
- `aria-disabled`
- `aria-label`
- `aria-level`

Finally, the following terms are defined in *ARIA*: [ARIA]p1579

- **role**
- The `ARIAMixin` interface, with its associated **`ARIAMixin` getter steps** and **`ARIAMixin` setter steps** hooks, and its `role` and `aria*` attributes

Accessible Name and Description Computation

The following term is defined in *Accessible Name and Description Computation*: [ACCNAME]p1579

- **accessible name**

Content Security Policy

The following terms are defined in *Content Security Policy*: [CSP]p1580

- **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 **Run CSP initialization for a Document** algorithm
- The **Run CSP initialization for a global object** 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.
- The **Parse a response's Content Security Policies** algorithm.
- `SecurityPolicyViolationEvent` interface
- The `securitypolicyviolation` event

Service Workers

The following terms are defined in *Service Workers*: [SW]p1587

- **active worker**
- **client message queue**
- **control**
- **handle fetch**
- **match service worker registration**
- **service worker**

- **service worker client**
- **service worker client**
- **service worker registration**
- `ServiceWorker` interface
- `ServiceWorkerContainer` interface
- `ServiceWorkerGlobalScope` interface
- `unregister`

Secure Contexts

The following algorithms are defined in *Secure Contexts*: [SECURE-CONTEXTS]p1586

- **Is url potentially trustworthy?**

Permissions Policy

The following terms are defined in *Permissions Policy*: [PERMISSIONSPOLICY]p1585

- **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]p1585

- `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]p1584

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

- **MathML** `a` element
- **MathML** `annotation-xml` element
- **MathML** `math` element
- **MathML** `merror` element
- **MathML** `mfrac` element
- **MathML** `mi` element
- **MathML** `mmultiscripts` element
- **MathML** `mn` element
- **MathML** `mo` element
- **MathML** `mover` element
- **MathML** `mpadded` element
- **MathML** `mphantom` element
- **MathML** `mprescripts` element
- **MathML** `mroot` element
- **MathML** `mrow` element
- **MathML** `ms` element
- **MathML** `mspace` element
- **MathML** `msqrt` element
- **MathML** `mstyle` element
- **MathML** `msub` element
- **MathML** `msubsup` element
- **MathML** `msup` element
- **MathML** `mtable` element
- **MathML** `mtd` element
- **MathML** `mtext` element
- **MathML** `mtr` element
- **MathML** `munder` element
- **MathML** `munderover` element
- **MathML** `semantics` element
- **MathML** `accent` attribute
- **MathML** `accentunder` attribute
- **MathML** `columnspan` attribute
- **MathML** `depth` attribute
- **MathML** `fence` attribute
- **MathML** `form` attribute
- **MathML** `height` attribute
- **MathML** `href` attribute
- **MathML** `hreflang` attribute
- **MathML** `largeop` attribute
- **MathML** `lspace` attribute
- **MathML** `maxsize` attribute
- **MathML** `minsize` attribute

- **MathML** `movablelimits` attribute
- **MathML** `rowspan` attribute
- **MathML** `rspace` attribute
- **MathML** `separator` attribute
- **MathML** `stretchy` attribute
- **MathML** `symmetric` attribute
- **MathML** `type` attribute
- **MathML** `voffset` attribute
- **MathML** `width` attribute
- **MathML** `displaystyle` attribute
- **MathML** `mathbackground` attribute
- **MathML** `mathcolor` attribute
- **MathML** `mathsize` attribute
- **MathML** `scriptlevel` attribute

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][p1586]

- `SVGElement` interface
- `SVGImageElement` interface
- `SVGScriptElement` interface
- `SVGSVGElement` interface
- **SVG** `a` element
- **SVG** `animate` element
- **SVG** `animateTransform` element
- **SVG** `circle` element
- **SVG** `defs` element
- **SVG** `desc` element
- **SVG** `ellipse` element
- **SVG** `foreignObject` element
- **SVG** `g` element
- **SVG** `image` element
- **SVG** `line` element
- **SVG** `marker` element
- **SVG** `metadata` element
- **SVG** `path` element
- **SVG** `polygon` element
- **SVG** `polyline` element
- **SVG** `rect` element
- **SVG** `script` element
- **SVG** `set` element
- **SVG** `svg` element
- **SVG** `text` element
- **SVG** `textPath` element
- **SVG** `title` element
- **SVG** `tspan` element
- **SVG** `use` element
- **SVG** `action` attribute
- **SVG** `attributeName` attribute
- **SVG** `cx` attribute
- **SVG** `cy` attribute
- **SVG** `d` attribute
- **SVG** `dx` attribute
- **SVG** `dy` attribute
- **SVG** `formaction` attribute
- **SVG** `height` attribute
- **SVG** `href` attribute
- **SVG** `hreflang` attribute
- **SVG** `lengthAdjust` attribute
- **SVG** `markerHeight` attribute
- **SVG** `markerUnits` attribute
- **SVG** `markerWidth` attribute
- **SVG** `method` attribute
- **SVG** `orient` attribute
- **SVG** `path` **attribute**
- **SVG** `pathLength` attribute
- **SVG** `points` attribute
- **SVG** `preserveAspectRatio` attribute
- **SVG** `r` attribute
- **SVG** `refX` attribute
- **SVG** `refY` attribute
- **SVG** `rotate` attribute
- **SVG** `rx` attribute
- **SVG** `ry` attribute
- **SVG** `side` attribute
- **SVG** `spacing` attribute

- **SVG `startOffset`** attribute
- **SVG `textLength`** attribute
- **SVG `type`** attribute
- **SVG `viewBox`** attribute
- **SVG `width`** attribute
- **SVG `x`** attribute
- **SVG `x1`** attribute
- **SVG `x2`** attribute
- **SVG `y`** attribute
- **SVG `y1`** attribute
- **SVG `y2`** attribute
- **SVG `text-rendering`** property
- **SVG `alignment-baseline`** property
- **SVG `baseline-shift`** property
- **SVG `clip-path`** property
- **SVG `clip-rule`** property
- **SVG `color`** property
- **SVG `color-interpolation`** property
- **SVG `cursor`** property
- **SVG `direction`** property
- **SVG `display`** property
- **SVG `dominant-baseline`** property
- **SVG `fill`** property
- **SVG `fill-opacity`** property
- **SVG `fill-rule`** property
- **SVG `font-family`** property
- **SVG `font-size`** property
- **SVG `font-size-adjust`** property
- **SVG `font-stretch`** property
- **SVG `font-style`** property
- **SVG `font-variant`** property
- **SVG `font-weight`** property
- **SVG `letter-spacing`** property
- **SVG `marker-end`** property
- **SVG `marker-mid`** property
- **SVG `marker-start`** property
- **SVG `opacity`** property
- **SVG `paint-order`** property
- **SVG `pointer-events`** property
- **SVG `shape-rendering`** property
- **SVG `stop-color`** property
- **SVG `stop-opacity`** property
- **SVG `stroke`** property
- **SVG `stroke-dasharray`** property
- **SVG `stroke-dashoffset`** property
- **SVG `stroke-linecap`** property
- **SVG `stroke-linejoin`** property
- **SVG `stroke-miterlimit`** property
- **SVG `stroke-opacity`** property
- **SVG `stroke-width`** property
- **SVG `text-anchor`** property
- **SVG `text-decoration`** property
- **SVG `text-overflow`** property
- **SVG `transform`** property
- **SVG `transform-origin`** property
- **SVG `unicode-bidi`** property
- **SVG `vector-effect`** property
- **SVG `visibility`** property
- **SVG `white-space`** property
- **SVG `word-spacing`** property
- **SVG `writing-mode`** property

Filter Effects

The following features are defined in *Filter Effects*: [FILTERS][p1582]

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

Compositing

The following features are defined in *Compositing and Blending*: [COMPOSITE][p1579]

- **<blend-mode>**
- **<composite-mode>**
- **source-over**
- **copy**

Cooperative Scheduling of Background Tasks

The following features are defined in *Cooperative Scheduling of Background Tasks*: [REQUESTIDLECALLBACK][p1585]

- **`requestIdleCallback()`**
- **start an idle period algorithm**

Screen Orientation

The following terms are defined in *Screen Orientation*: [SCREENORIENTATION][p1586]

- **screen orientation change steps**

Storage

The following terms are defined in *Storage*: [STORAGE][p1586]

- **storage key**
- **obtain a local storage bottle map**
- **obtain a session storage bottle map**
- **obtain a storage key for non-storage purposes**
- **storage key equal**
- **storage proxy map**
- **legacy-clone a traversable storage shed**

Web App Manifest

The following features are defined in *Web App Manifest*: [MANIFEST][p1584]

- **application manifest**
- **installed web application**
- **process the manifest**

WebAssembly JavaScript Interface: ESM Integration

The following terms are defined in *WebAssembly JavaScript Interface: ESM Integration*: [WASMESM][p1587]

- **WebAssembly Module Record**
- **parse a WebAssembly module**

WebCodecs

The following features are defined in *WebCodecs*: [WEBCODECS][p1588]

- `VideoFrame` interface.
- **[[display width]]**
- **[[display height]]**

WebDriver

The following terms are defined in *WebDriver*: [WEBDRIVER][p1588]

- **extension command**
- **remote end steps**
- **WebDriver error**
- **WebDriver error code**
- **invalid argument**
- **getting a property**
- **success**
- **WebDriver's security considerations**
- **current browsing context**

WebDriver BiDi

The following terms are defined in *WebDriver BiDi*: [WEBDRIVERBIDI][p1588]

- **WebDriver BiDi navigation status**
- **navigation status id**
- **navigation status status**
- **navigation status canceled**
- **navigation status committed**
- **navigation status pending**
- **navigation status complete**
- **navigation status url**
- **navigation status suggested filename**
- **download behavior allowed**
- **download behavior destination folder**
- **navigation status downloaded filepath**
- **WebDriver BiDi navigation aborted**
- **WebDriver BiDi navigation committed**
- **WebDriver BiDi navigation failed**
- **WebDriver BiDi navigation started**
- **WebDriver BiDi download end**
- **WebDriver BiDi download will begin**
- **WebDriver BiDi fragment navigated**
- **WebDriver BiDi DOM content loaded**
- **WebDriver BiDi load complete**
- **WebDriver BiDi history updated**
- **WebDriver BiDi navigable created**
- **WebDriver BiDi navigable destroyed**

- **WebDriver BiDi user prompt closed**
- **WebDriver BiDi user prompt opened**
- **WebDriver BiDi file dialog opened**
- **WebDriver BiDi emulated language**
- **WebDriver BiDi scripting is enabled**

Web Cryptography API

The following terms are defined in *Web Cryptography API*: [WEBCRYPTO][p1588]

- **generating a random UUID**

WebSockets

The following terms are defined in *WebSockets*: [WEBSOCKETS][p1588]

- `WebSocket`
- **make disappear**

WebTransport

The following terms are defined in *WebTransport*: [WEBTRANSPORT][p1588]

- `WebTransport`
- `context cleanup steps`

Web Authentication: An API for accessing Public Key Credentials

The following terms are defined in *Web Authentication: An API for accessing Public Key Credentials*: [WEBAUTHN][p1587]

- **public key credential**

Credential Management

The following terms are defined in *Credential Management*: [CREDMAN][p1580]

- **conditional mediation**
- **credential**
- `navigator.credentials.get()`

Console

The following terms are defined in *Console*: [CONSOLE][p1580]

- **report a warning to the console**

Web Locks API

The following terms are defined in *Web Locks API*: [WEBLOCKS][p1588]

- **locks**
- **lock requests**

Trusted Types

This specification uses the following features defined in *Trusted Types*: [TRUSTED-TYPES][p1587]

- `TrustedHTML`
- **data**
- `TrustedScript`
- `data`
- `TrustedScriptURL`
- **get trusted type compliant string**
- The `require-trusted-types-for` **pre-navigation check** algorithm

WebRTC API

The following terms are defined in *WebRTC API*: [WEBRTC][p1588]

- `RTCDataChannel`
- `RTCPeerConnection`

Picture-in-Picture API

The following terms are defined in *Picture-in-Picture API*: [PICTUREINPICTURE][p1585]

- `PictureInPictureWindow`

Idle Detection API

The following terms are defined in *Idle Detection API*:

- `IdleDetector`

Web Speech API

The following terms are defined in *Web Speech API*:

- `SpeechRecognition`

WebOTP API

The following terms are defined in *WebOTP API*:

- `OTPCredential`

Web Share API

The following terms are defined in *Web Share API*:

- **share()**

Web Smart Card API

The following terms are defined in *Web Smart Card API*:

- `SmartCardConnection`

Web Background Synchronization

The following terms are defined in *Web Background Synchronization*:

- `SyncManager`
- `register()`

Web Periodic Background Synchronization

The following terms are defined in *Web Periodic Background Synchronization*:

- `PeriodicSyncManager`
- `register()`

Web Background Fetch

The following terms are defined in *Background Fetch*:

- `BackgroundFetchManager`
- `fetch()`

Keyboard Lock

The following terms are defined in *Keyboard Lock*:

- `Keyboard`
- `lock()`

Web MIDI API

The following terms are defined in *Web MIDI API*:

- `requestMIDIAccess()`

Generic Sensor API

The following terms are defined in *Generic Sensor API*:

- **request sensor access**

WebHID API

The following terms are defined in *WebHID API*:

- `requestDevice`

WebXR Device API

The following terms are defined in *WebXR Device API*:

- `XRSystem`

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] [COOKIES]

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

Avoid using client-side XSLT in browsers. The browser implementations of XSLT are highly susceptible to memory safety vulnerabilities, and the web platform has evolved toward JavaScript as the safe replacement. For these reasons, this feature is in the process of being removed from the web platform. (This is a long process that takes many years.)

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][p1589]

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][p1589]

This specification does not specify precisely how XSLT processing interacts with the HTML parser[p1368] infrastructure (for example, whether an XSLT processor acts as if it puts any elements into a stack of open elements[p1383]). However, XSLT processors must stop parsing[p1458] if they successfully complete, and must update the current document readiness[p141] first to "`interactive`" and then to "`complete`" if they are aborted.

This specification does not specify how XSLT interacts with the navigation[p1063] algorithm, how it fits in with the event loop[p1193], 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.).

Note

There are also additional non-normative comments regarding the interaction of XSLT and HTML in the `script` element section[p702]*, and of XSLT, XPath, and HTML in the `template` element section*[p711].

2.2 Policy-controlled features §p79

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'`.
- "**focus-without-user-activation**", which has a default allowlist of `'self'`.

2.3 Common microsyntaxes §p79

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

Implementers 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 §p79

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 §p79

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[p543] and disabled[p630] attributes are the boolean attributes.

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

This could be equivalently written as this:

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

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

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

2.3.3 Keywords and enumerated attributes §[p80]

Some attributes, called **enumerated attributes**, take on a finite set of states. The state for such an attribute is derived by combining the attribute's value, a set of keyword/state mappings, and three possible special states that can also be given in the specification of the attribute. These special states are the ***invalid value default***, the ***missing value default***, and the ***empty value default***.

Note

Multiple keywords can map to the same state.

To determine the state of an attribute, use the following steps:

1. If the attribute is not specified:
    1. If the attribute has a *missing value default*[p80] state defined, then return that *missing value default*[p80] state.
    2. Otherwise, return no state.
2. If the attribute's value is an ASCII case-insensitive match for one of the keywords defined for the attribute, then return the state represented by that keyword.
3. If the attribute has an *empty value default*[p80] state defined and the attribute's value is the empty string, then return that *empty value default*[p80] state.
4. If the attribute has an *invalid value default*[p80] state defined, then return that *invalid value default*[p80] state.
5. Return no state.

For authoring conformance purposes, if an enumerated attribute is specified, the attribute's value must be one of:

- An ASCII case-insensitive match for one of the conforming keywords for that attribute, with no leading or trailing whitespace.
- The empty string and the attribute must have an *empty value default*[p80] defined.

For reflection[p108] purposes, states which have any keywords mapping to them are said to have a **canonical keyword**. This is determined as follows:

1. If there is only one keyword mapping to the given state, then it is that keyword.
2. If there is only one *conforming* keyword mapping to the given state, then it is that conforming keyword.
3. If there are two conforming keywords mapping to the given state, and one is the empty string, then the canonical keyword will be the conforming keyword that is not the empty string.
4. Otherwise, the canonical keyword for the state will be explicitly given in the specification for the attribute.

2.3.4 Numbers §[p80]

2.3.4.1 Signed integers §[p80]

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[p80] without a U+002D HYPHEN-MINUS (-) prefix represents the number that is represented in base ten by that string of digits. A valid integer[p80] *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 *position* (the first character) is a U+002D HYPHEN-MINUS character (-):
    1. Let *sign* be "negative".
    2. Advance *position* to the next character.
    3. If *position* is past the end of *input*, return an error.

    Otherwise, if the character indicated by *position* (the first character) is a U+002B PLUS SIGN character (+):
    1. Advance *position* to the next character. (The "+" is ignored, but it is not conforming.)
    2. If *position* is past the end of *input*, return an error.
7. If the character indicated by *position* is not an ASCII digit, then return an error.
8. Collect a sequence of code points that are ASCII digits from *input* given *position*, and interpret the resulting sequence as a base-ten integer. Let *value* be that integer.
9. If *sign* is "positive", return *value*, otherwise return the result of subtracting *value* from zero.

2.3.4.2 Non-negative integers §[p81]

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

A valid non-negative integer[p81] represents the number that is represented in base ten by that string of digits.

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

1. Let *input* be the string being parsed.
2. Let *value* be the result of parsing *input* using the rules for parsing integers[p80].
3. If *value* is an error, return an error.
4. If *value* is less than zero, return an error.
5. Return *value*.

2.3.4.3 Floating-point numbers §[p81]

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[p81] represents the number obtained by multiplying the significand by ten raised to the power of the

exponent, where the significand is the first number, interpreted as base ten (including the decimal point and the number after the decimal point, if any, and interpreting the significand as a negative number if the whole string starts with a U+002D HYPHEN-MINUS character (-) and the number is not zero), and where the exponent is the number after the E, if any (interpreted as a negative number if there is a U+002D HYPHEN-MINUS character (-) between the E and the number and the number is not zero, or else ignoring a U+002B PLUS SIGN character (+) between the E and the number if there is one). If there is no E, then the exponent is treated as zero.

Note

The Infinity and Not-a-Number (NaN) values are not valid floating-point numbers[p81].

Note

The valid floating-point number[p81] concept is typically only used to restrict what is allowed for authors, while the user agent requirements use the rules for parsing floating-point number values[p82] below (e.g., the `max`[p614] attribute of the `progress`[p614] element). However, in some cases the user agent requirements include checking if a string is a valid floating-point number[p81] (e.g., the value sanitization algorithm[p543] for the Number[p556] state of the `input`[p539] element, or the parse a srcset attribute[p387] algorithm).

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

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

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

    Otherwise, if the character indicated by *position* (the first character) is a U+002B PLUS SIGN character (+):
    1. Advance *position* to the next character. (The "+" is ignored, but it is not conforming.)
    2. If *position* is past the end of *input*, return an error.
9. If the character indicated by *position* is a U+002E FULL STOP (.), and that is not the last character in *input*, and the character after the character indicated by *position* is an ASCII digit, then set *value* to zero and jump to the step labeled *fraction*.
10. If the character indicated by *position* is not an ASCII digit, then return an error.
11. Collect a sequence of code points that are ASCII digits from *input* given *position*, and interpret the resulting sequence as a base-ten integer. Multiply *value* by that integer.
12. If *position* is past the end of *input*, jump to the step labeled *conversion*.
13. *Fraction*: If the character indicated by *position* is a U+002E FULL STOP (.), run these substeps:
    1. Advance *position* to the next character.
    2. If *position* is past the end of *input*, or if the character indicated by *position* is not an ASCII digit, U+0065 LATIN SMALL LETTER E (e), or U+0045 LATIN CAPITAL LETTER E (E), then jump to the step labeled *conversion*.

3. If the character indicated by *position* is a U+0065 LATIN SMALL LETTER E character (e) or a U+0045 LATIN CAPITAL LETTER E character (E), skip the remainder of these substeps.
4. *Fraction loop*: Multiply *divisor* by ten.
5. Add the value of the character indicated by *position*, interpreted as a base-ten digit (0..9) and divided by *divisor*, to *value*.
6. Advance *position* to the next character.
7. If *position* is past the end of *input*, then jump to the step labeled *conversion*.
8. If the character indicated by *position* is an ASCII digit, jump back to the step labeled *fraction loop* in these substeps.

14. If the character indicated by *position* is U+0065 (e) or a U+0045 (E):
    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 *exponent*th power.
15. *Conversion*: Let *S* be the set of finite IEEE 754 double-precision floating-point values except −0, but with two special values added: 2^{1024} and -2^{1024}.
16. Let *rounded-value* be the number in *S* that is closest to *value*, selecting the number with an even significand if there are two equally close values. (The two special values 2^{1024} and -2^{1024} are considered to have even significands for this purpose.)
17. If *rounded-value* is 2^{1024} or -2^{1024}, return an error.
18. Return *rounded-value*.

2.3.4.4 Percentages and lengths §[p83]

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 (.):
    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[p84] 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[p84] 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 Nonzero percentages and lengths §[p84]

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[p83].
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 §[p84]

A **valid list of floating-point numbers** is a number of valid floating-point numbers[p81] 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[p82].

    4. If *number* is an error, set *number* to zero.

    5. Append *number* to *numbers*.

    6. 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 the delimiter.

6. Return *numbers*.

2.3.4.7 Lists of dimensions §[p85]

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

1. Let *raw input* be the string being parsed.

2. If the last character in *raw input* is a U+002C COMMA character (,), then remove that character from *raw input*.

3. Split the string *raw input* on commas. Let *raw tokens* be the resulting list of tokens.

4. Let *result* be an empty list of number/unit pairs.

5. For each token in *raw tokens*, run the following substeps:

    1. Let *input* be the token.

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

    3. Let *value* be the number 0.

    4. Let *unit* be *absolute*.

    5. If *position* is past the end of *input*, set *unit* to *relative* and jump to the last substep.

    6. If the character at *position* is an ASCII digit, collect a sequence of code points that are ASCII digits from *input* given *position*, interpret the resulting sequence as an integer in base ten, and increment *value* by that integer.

    7. If the character at *position* is U+002E (.):

        1. Collect a sequence of code points consisting of ASCII whitespace and ASCII digits from *input* given *position*. Let *s* be the resulting sequence.

        2. Remove all ASCII whitespace in *s*.

        3. If *s* is not the empty string:

            1. Let *length* be the number of characters in *s* (after the spaces were removed).

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

            3. Increment *value* by *fraction*.

    8. Skip ASCII whitespace within *input* given *position*.

9. If the character at *position* is a U+0025 PERCENT SIGN character (%), then set *unit* to *percentage*.

   Otherwise, if the character at *position* is a U+002A ASTERISK character (*), then set *unit* to *relative*.

10. Add an entry to *result* consisting of the number given by *value* and the unit given by *unit*.

6. Return the list *result*.

2.3.5 Dates and times §p86

In the algorithms below, the **number of days in month *month* of year *year*** is: *31* if *month* is 1, 3, 5, 7, 8, 10, or 12; *30* if *month* is 4, 6, 9, or 11; *29* if *month* is 2 and *year* is a number divisible by 400, or if *year* is a number divisible by 4 but not by 100; and *28* otherwise. This takes into account leap years in the Gregorian calendar. [GREGORIAN]p1582

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. Implementers are therefore encouraged to carefully examine any date parsing libraries before using them to implement the parsing rules described below; ISO8601 libraries might not parse dates and times in exactly the same manner. [ISO8601]p1583

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 calendarp86, sometimes explicitly referred to as a **proleptic-Gregorian date**, is one that is described using that calendar even if that calendar was not in use at the time (or place) in question. [GREGORIAN]p1582

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 formatsp531 in forms (for authors), implementation notes regarding localization of form controlsp569, and the `time`p290 element.

2.3.5.1 Months §p86

A **month** consists of a specific proleptic-Gregorian datep86 with no time-zone information and no date information beyond a year and a month. [GREGORIAN]p1582

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

1. Four or more ASCII digits, representing *year*, where *year* > 0
2. A U+002D HYPHEN-MINUS character (-)
3. Two ASCII digits, representing the month *month*, in the range 1 ≤ *month* ≤ 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 componentp86 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 *year* be that number.
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 *month* be that number.
5. If *month* is not a number in the range $1 \leq month \leq 12$, then fail.
6. Return *year* and *month*.

2.3.5.2 Dates §p87

A **date** consists of a specific proleptic-Gregorian date[p86] with no time-zone information, consisting of a year, a month, and a day. [GREGORIAN][p1582]

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

1. A valid month string[p86], representing *year* and *month*
2. A U+002D HYPHEN-MINUS character (-)
3. Two ASCII digits, representing *day*, in the range $1 \leq day \leq maxday$ where *maxday* is the number of days in the month *month* and year *year*[p86]

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[p87] to obtain *year*, *month*, and *day*. If this returns nothing, then fail.
4. If *position* is *not* beyond the end of *input*, then fail.
5. Let *date* be the date with year *year*, month *month*, and day *day*.
6. Return *date*.

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

1. Parse a month component[p86] to obtain *year* and *month*. If this returns nothing, then fail.
2. Let *maxday* be the number of days in month *month* of year *year*[p86].
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 *day* be that number.
5. If *day* is not a number in the range $1 \leq day \leq maxday$, then fail.
6. Return *year*, *month*, and *day*.

2.3.5.3 Yearless dates §p87

A **yearless date** consists of a Gregorian month and a day within that month, but with no associated year. [GREGORIAN][p1582]

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

1. Optionally, two U+002D HYPHEN-MINUS characters (-)
2. Two ASCII digits, representing the month *month*, in the range 1 ≤ *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[p86] in the month *month* and any arbitrary leap year (e.g. 4 or 2000)

Note

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

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[p88] 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 *month* be that number.
3. If *month* is not a number in the range 1 ≤ *month* ≤ 12, then fail.
4. Let *maxday* be the number of days[p86] in month *month* of any arbitrary leap year (e.g. 4 or 2000).
5. If *position* is beyond the end of *input* or if the character at *position* is not a U+002D HYPHEN-MINUS character, then fail. Otherwise, move *position* forwards one character.
6. Collect a sequence of 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 *day* be that number.
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 §[p88]

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

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

1. Two ASCII digits, representing *hour*, in the range 0 ≤ *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 $0 \le s \le 59$
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[p89] 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 *hour* be that number.
2. If *hour* is not a number in the range $0 \le hour \le 23$, 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 *minute* be that number.
5. If *minute* is not a number in the range $0 \le minute \le 59$, 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 (:):
    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 $0 \le second < 60$, then fail.
8. Return *hour*, *minute*, and *second*.

2.3.5.5 Local dates and times §[p89]

A **local date and time** consists of a specific proleptic-Gregorian date[p86], consisting of a year, a month, and a day, and a time, consisting of an hour, a minute, a second, and a fraction of a second, but expressed without a time zone. [GREGORIAN][p1582]

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[p87] representing the date
2. A U+0054 LATIN CAPITAL LETTER T character (T) or a U+0020 SPACE character
3. A valid time string[p88] 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[p87] representing the date
2. A U+0054 LATIN CAPITAL LETTER T character (T)
3. A valid time string[p88] 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[p87] 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[p89] to obtain *hour*, *minute*, and *second*. If this returns nothing, then fail.
6. If *position* is *not* beyond the end of *input*, then fail.
7. Let *date* be the date with year *year*, month *month*, and day *day*.
8. Let *time* be the time with hour *hour*, minute *minute*, and second *second*.
9. Return *date* and *time*.

2.3.5.6 Time zones §[p90]

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 $0 \le hour \le 23$
    3. Optionally, a U+003A COLON character (:)
    4. Two ASCII digits, representing the minutes component *minute* of the time-zone offset, in the range $0 \le minute \le 59$

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[p91] 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[p91] 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):
    1. Let $timezone_{hours}$ be 0.
    2. Let $timezone_{minutes}$ be 0.
    3. Advance *position* to the next character in *input*.

    Otherwise, if the character at *position* is either a U+002B PLUS SIGN (+) or a U+002D HYPHEN-MINUS (-):
    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:
        1. Interpret *s* as a base-ten integer. Let $timezone_{hours}$ be that number.
        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 $timezone_{minutes}$ be that number.

        If *s* is exactly four characters long:
        1. Interpret the first two characters of *s* as a base-ten integer. Let $timezone_{hours}$ be that number.
        2. Interpret the last two characters of *s* as a base-ten integer. Let $timezone_{minutes}$ be that number.

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

    Otherwise, fail.
2. Return $timezone_{hours}$ and $timezone_{minutes}$.

2.3.5.7 Global dates and times §[p91]

A **global date and time** consists of a specific proleptic-Gregorian date[p86], 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][p1582]

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

1. A valid date string[p87] representing the date
2. A U+0054 LATIN CAPITAL LETTER T character (T) or a U+0020 SPACE character
3. A valid time string[p88] representing the time
4. A valid time-zone offset string[p90] 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[p92].

"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[p86].
- The time and time-zone offset components are not optional.
- Dates before the year one can't be represented as a datetime in this version of HTML.
- Times of specific events in ancient times are, at best, approximations, since time was not well coordinated or measured until relatively recent decades.
- Time-zone offsets differ based on daylight 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 roundtripping 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[p87] 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[p89] 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[p91] to obtain $timezone_{hours}$ and $timezone_{minutes}$. If this returns nothing, then fail.
8. If *position* is *not* beyond the end of *input*, then fail.
9. Let *time* be the moment in time at year *year*, month *month*, day *day*, hours *hour*, minute *minute*, second *second*, subtracting $timezone_{hours}$ hours and $timezone_{minutes}$ minutes. That moment in time is a moment in the UTC time zone.
10. Let *timezone* be $timezone_{hours}$ hours and $timezone_{minutes}$ minutes from UTC.
11. Return *time* and *timezone*.

2.3.5.8 Weeks §[p93]

A **week** consists of a week-year number and a week number representing a seven-day period starting on a Monday. Each week-year in this calendaring system has either 52 or 53 such seven-day periods, as defined below. The seven-day period starting on the Gregorian date Monday December 29th 1969 (1969-12-29) is defined as week number 1 in week-year 1970. Consecutive weeks are numbered sequentially. The week before the number 1 week in a week-year is the last week in the previous week-year, and vice versa. [GREGORIAN][p1582]

A week-year with a number *year* has 53 weeks if it corresponds to either a year *year* in the proleptic Gregorian calendar[p86] that has a Thursday as its first day (January 1st), or a year *year* in the proleptic Gregorian calendar[p86] that has a Wednesday as its first day (January 1st) and where *year* is a number divisible by 400, or a number divisible by 4 but not by 100. All other week-years have 52 weeks.

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

Note

The week-year number of a particular day can be different than the number of the year that contains that day in the proleptic Gregorian calendar[p86]. The first week in a week-year y is the week that contains the first Thursday of the Gregorian year y.

Note

For modern purposes, a week[p93] as defined here is equivalent to ISO weeks as defined in ISO 8601. [ISO8601][p1583]

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

1. Four or more ASCII digits, representing *year*, where $year > 0$
2. A U+002D HYPHEN-MINUS character (-)
3. A U+0057 LATIN CAPITAL LETTER W character (W)
4. Two ASCII digits, representing the week *week*, in the range $1 \leq week \leq maxweek$, where *maxweek* is the week number of the last day[p93] of week-year *year*

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

1. Let *input* be the string being parsed.
2. Let *position* be a pointer into *input*, initially pointing at the start of the string.
3. Collect a sequence of 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 *year* be that number.
4. If *year* is not a number greater than zero, then fail.
5. If *position* is beyond the end of *input* or if the character at *position* is not a U+002D HYPHEN-MINUS character, then fail. Otherwise, move *position* forwards one character.

6. If *position* is beyond the end of *input* or if the character at *position* is not a U+0057 LATIN CAPITAL LETTER W character (W), then fail. Otherwise, move *position* forwards one character.
7. Collect a sequence of 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 *week* be that number.
8. Let *maxweek* be the week number of the last day[p93] 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*.

2.3.5.9 Durations §[p94]

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[p94] 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[p94] *t* if it consists of either of the following:

- A literal U+0050 LATIN CAPITAL LETTER P character followed by one or more of the following subcomponents, in the order given, where the number of days, hours, minutes, and seconds corresponds to the same number of seconds as in *t*:
    1. One or more ASCII digits followed by a U+0044 LATIN CAPITAL LETTER D character, representing a number of days.
    2. A U+0054 LATIN CAPITAL LETTER T character followed by one or more of the following subcomponents, in the order given:
        1. One or more ASCII digits followed by a U+0048 LATIN CAPITAL LETTER H character, representing a number of hours.
        2. One or more ASCII digits followed by a U+004D LATIN CAPITAL LETTER M character, representing a number of minutes.
        3. The following components:
            1. One or more ASCII digits, representing a number of seconds.
            2. Optionally, a U+002E FULL STOP character (.) followed by one, two, or three ASCII digits, 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][p1583]

- One or more duration time components[p94], each with a different duration time component scale[p95], 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[p95] specified (see below) to represent a number of seconds.
    3. If the duration time component scale[p95] 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[p94]:

   U+0057 LATIN CAPITAL LETTER W character
   U+0077 LATIN SMALL LETTER W character
   Weeks. The scale is 604800.

   U+0044 LATIN CAPITAL LETTER D character
   U+0064 LATIN SMALL LETTER D character
   Days. The scale is 86400.

   U+0048 LATIN CAPITAL LETTER H character
   U+0068 LATIN SMALL LETTER H character
   Hours. The scale is 3600.

   U+004D LATIN CAPITAL LETTER M character
   U+006D LATIN SMALL LETTER M character
   Minutes. The scale is 60.

   U+0053 LATIN CAPITAL LETTER S character
   U+0073 LATIN SMALL LETTER S character
   Seconds. The scale is 1.

6. Zero or more 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[p94] 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* be 0. (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 (.):

   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[p94] consisting of *seconds* seconds.

2.3.5.10 Vaguer moments in time §[p97]

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

- A valid date string[p87]
- A valid global date and time string[p92]

The rules to **parse a date or time string** are as follows. The algorithm will return either a date[p87], a time[p88], a global date and time[p91], 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[p87] 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[p89] 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[p91] 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 Legacy colors §[p98]

Some obsolete legacy attributes parse colors using the **rules for parsing a legacy color value**, given a string *input*. They will return either a CSS color or failure.

1. If *input* is the empty string, then return failure.
2. Strip leading and trailing ASCII whitespace from *input*.
3. If *input* is an ASCII case-insensitive match for "`transparent`", then return failure.
4. If *input* is an ASCII case-insensitive match for one of the named colors, then return the CSS color corresponding to that keyword. [CSSCOLOR][p1580]

   Note

   CSS2 System Colors are not recognized.

5. 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:
   1. Let *result* be a CSS 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*.
6. Replace any code points greater than U+FFFF in *input* (i.e., any characters that are not in the basic multilingual plane) with "`00`".
7. If *input*'s code point length is greater than 128, truncate *input*, leaving only the first 128 characters.
8. If the first character in *input* is U+0023 (#), then remove it.
9. Replace any character in *input* that is not an ASCII hex digit with U+0030 (0).
10. While *input*'s code point length is zero or not a multiple of three, append U+0030 (0) to *input*.
11. 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*).
12. If *length* is greater than 8, then remove the leading *length*-8 characters in each component, and let *length* be 8.
13. While *length* is greater than two and the first character in each component is U+0030 (0), remove that character and reduce *length* by one.
14. If *length* is *still* greater than two, truncate each component, leaving only the first two characters in each.
15. Let *result* be a CSS color.
16. Interpret the first component as a hexadecimal number; let the red component of *result* be the resulting number.
17. Interpret the second component as a hexadecimal number; let the green component of *result* be the resulting number.
18. Interpret the third component as a hexadecimal number; let the blue component of *result* be the resulting number.
19. Return *result*.

2.3.7 Space-separated tokens §p99

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[p99] may have leading or trailing ASCII whitespace.

An **unordered set of unique space-separated tokens** is a set of space-separated tokens[p99] where none of the tokens are duplicated.

An **ordered set of unique space-separated tokens** is a set of space-separated tokens[p99] where none of the tokens are duplicated but where the order of the tokens is meaningful.

Sets of space-separated tokens[p99] 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[p99] are to be compared (e.g. case-sensitively or not) is defined on a per-set basis.

2.3.8 Comma-separated tokens §p99

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.

Example

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

Sets of comma-separated tokens[p99] 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 §p99

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`[p162] or `name` attribute whose value is *s*, or null if there is no such element.

Note

Although `id`[p162] attributes are accounted for when parsing, they are not used in determining whether a value is a valid hash-name reference[p99]. That is, a hash-name reference that refers to an element based on `id`[p162] is a conformance error (unless that element also has a `name` attribute with the same value).

2.3.10 Media queries §p10 0

A string is a **valid media query list** if it matches the `<media-query-list>` production of *Media Queries*. [MQ]p1584

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]p1584

2.3.11 Unique internal values §p10 0

A **unique internal value** is a value that is serializable, comparable by value, and never exposed to script.

To create a **new unique internal value**, return a unique internal valuep100 that has never previously been returned by this algorithm.

2.4 URLs §p10 0

2.4.1 Terminology §p10 0

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 URLp100.

This specification defines the URL **about:legacy-compat** as a reserved, though unresolvable, about: URL, for use in DOCTYPEp1356s in HTML documents when needed for compatibility with XML tools. [ABOUT]p1579

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]p1579

This specification defines the URL **about:srcdoc** as a reserved, though unresolvable, about: URL, that is used as the URL of iframe srcdoc documentsp405. [ABOUT]p1579

A URL **matches about:blank** if its scheme is "about", its path contains a single string "blank", its username and password are the empty string, and its host is null.

Note

Such a URL's query and fragment can be non-null. For example, the URL record created by parsing "about:blank?foo#bar" matches about:blankp100.

A URL **matches about:srcdoc** if its scheme is "about", its path contains a single string "srcdoc", its query is null, its username and password are the empty string, and its host is null.

Note

The reason that matches about:srcdocp100 ensures that the URL's query is null is because it is not possible to create an iframe srcdoc documentp405 whose URL has a non-null query, unlike Documentp135s whose URL matches about:blankp100. In other words, the set of all URLs that match about:srcdocp100 only vary in their fragment.

2.4.2 Parsing URLs §p10 0

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 several wrappers to abstract base URLs and encodings. [URL]p1587

Note

Most new APIs are to use parse a URL[p101]. Older APIs and HTML elements might have reason to use encoding-parse a URL[p101]. When a custom base URL is needed or no base URL is desired, the URL parser can of course be used directly as well.

To **parse a URL**, given a string *url*, relative to a `Document`[p135] object or environment settings object[p1144] *environment*, run these steps. They return failure or a URL.

1. Let *baseURL* be *environment*'s base URL[p101], if *environment* is a `Document`[p135] object; otherwise *environment*'s API base URL[p1144].
2. Return the result of applying the URL parser to *url*, with *baseURL*.

To **encoding-parse a URL**, given a string *url*, relative to a `Document`[p135] object or environment settings object[p1144] *environment*, run these steps. They return failure or a URL.

1. Let *encoding* be UTF-8.
2. If *environment* is a `Document`[p135] object, then set *encoding* to *environment*'s character encoding.
3. Otherwise, if *environment*'s relevant global object[p1151] is a `Window`[p965] object, set *encoding* to *environment*'s relevant global object[p1151]'s associated `Document`[p966]'s character encoding.
4. Let *baseURL* be *environment*'s base URL[p101], if *environment* is a `Document`[p135] object; otherwise *environment*'s API base URL[p1144].
5. Return the result of applying the URL parser to *url*, with *baseURL* and *encoding*.

To **encoding-parse-and-serialize a URL**, given a string *url*, relative to a `Document`[p135] object or environment settings object[p1144] *environment*, run these steps. They return failure or a string.

1. Let *url* be the result of encoding-parsing a URL[p101] given *url*, relative to *environment*.
2. If *url* is failure, then return failure.
3. Return the result of applying the URL serializer to *url*.

2.4.3 Document base URLs §[p101]

The **document base URL** of a `Document`[p135] *document* is the URL record obtained by running these steps:

1. If *document* has no descendant `base`[p182] element that has an `href`[p183] attribute, then return *document*'s fallback base URL[p101].
2. Otherwise, return the frozen base URL[p183] of the first `base`[p182] element in *document* that has an `href`[p183] attribute, in tree order.

The **fallback base URL** of a `Document`[p135] object *document* is the URL record obtained by running these steps:

1. If *document* is an `iframe srcdoc` document[p405]:
    1. Assert: *document*'s about base URL[p136] is non-null.
    2. Return *document*'s about base URL[p136].
2. If *document*'s URL matches `about:blank`[p100] and *document*'s about base URL[p136] is non-null, then return *document*'s about base URL[p136].
3. Return *document*'s URL.

To **set the URL** for a `Document`[p135] *document* to a URL record *url*:

1. Set *document*'s URL to *url*.
2. Respond to base URL changes[p102] given *document*.

To **respond to base URL changes** for a Document[p135] *document*:

1. The user agent should update any user interface elements which are displaying affected URLs, or data derived from such URLs, to the user. Examples of such user interface elements would be a status bar that displays a hyperlink[p313]'s url[p316], or some user interface which displays the URL specified by a q[p277], blockquote[p244], ins[p350], or del[p351] element's `cite` attribute.
2. Ensure that the CSS :link[p821]/:visited[p821]/etc. pseudo-classes are updated appropriately.
3. For each *descendant* of *document*'s shadow-including descendants:
    1. If *descendant* is a script[p685] element whose result[p692] is a speculation rules parse result[p1170]:
        1. Let *oldResult* be *element*'s result[p692].
        2. Let *newResult* be the result of creating a speculation rules parse result[p1170] given *element*'s child text content and *element*'s node document.
        3. Update speculation rules[p1171] given *element*'s relevant global object[p1151], *oldResult*, and *newResult*.
4. Consider speculative loads[p1128] given *document*.

¶[p10] 2 **Example**

This means that changing the base URL doesn't affect, for example, the image displayed by img[p359] elements. Thus, subsequent accesses of the src[p360] IDL attribute from script will return a new absolute URL that might no longer correspond to the image being shown.

2.5 Fetching resources §[p10] 2

2.5.1 Terminology §[p10] 2

A response whose type is "`basic`", "`cors`", or "`default`" is **CORS-same-origin**. [FETCH][p1582]

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[p103], 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[p103], set *credentialsMode* to "`same-origin`".
5. Return 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 §[p10] 2

The **Content-Type metadata** of a resource must be obtained and interpreted in a manner consistent with the requirements of *MIME Sniffing*. [MIMESNIFF][p1584]

The **computed MIME type** of a resource must be found in a manner consistent with the requirements given in *MIME Sniffing*. [MIMESNIFF][p1584]

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][p1584]

⚠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][p1584]

2.5.3 Extracting character encodings from `meta`[p196] elements §[p10]

The **algorithm for extracting a character encoding from a `meta` element**, given a string *s*, is as follows. It returns either a character encoding or nothing.

1. Let *position* be a pointer into *s*, initially pointing at the start of the string.
2. *Loop*: Find the first seven characters in *s* after *position* that are an ASCII case-insensitive match for the word "`charset`". If no such match is found, return nothing.
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.

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][p1583]

✔ MDN

2.5.4 CORS settings attributes §[p10]

A **CORS settings attribute** is an enumerated attribute[p80] with the following keywords and states:

Keyword	State	Brief description
`anonymous`	**Anonymous**	Requests for the element will have their mode set to "`cors`" and their credentials mode set to "`same-origin`".
`use-credentials`	**Use Credentials**	Requests for the element will have their mode set to "`cors`" and their credentials mode set to "`include`".

The attribute's *missing value default*[p80] is the **No CORS** state, and its *invalid value default*[p80] and *empty value default*[p80] are both the Anonymous[p103] state.

The majority of fetches governed by CORS settings attributes[p103] will be done via the create a potential-CORS request[p102] algorithm.

For more modern features, where the request's mode is always "`cors`", certain CORS settings attributes[p103] 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[p103] to be determined by switching on the attribute's state:

↪ **No CORS**[p103]

↪ **Anonymous**[p103]

"same-origin"

↪ **Use Credentials**[p103]

"include"

2.5.5 Referrer policy attributes §[p104]

A **referrer policy attribute** is an enumerated attribute[p80]. Each referrer policy, including the empty string, is a keyword for this attribute, mapping to a state of the same name.

The attribute's *missing value default*[p80] and *invalid value default*[p80] 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][p1582] [REFERRERPOLICY][p1585]

Note

Several signals can contribute to which processing model is used for a given fetch; a referrer policy attribute[p104] *is only one of them. In general, the order in which these signals are processed are:*

1. *First, the presence of a* `noreferrer`[p338] *link type;*
2. *Then, the value of a referrer policy attribute*[p104]*;*
3. *Then, the presence of any* `meta`[p196] *element with* `name`[p197] *attribute set to* `referrer`[p199]*.*
4. *Finally, the* `Referrer-Policy` *HTTP header.*

2.5.6 Nonce attributes §[p104]

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][p1580]

Elements that have a `nonce`[p104] 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 `HTMLOrSVGOrMathMLElement`[p150] 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)

***element*.nonce**[p104]

Returns the value set for *element*'s cryptographic nonce. If the setter was not used, this will be the value originally found in the `nonce`[p104] content attribute.

element*.nonce**[p104] **= *value

Updates *element*'s cryptographic nonce value.

The **nonce** IDL attribute must, on getting, return the value of this element's [[CryptographicNonce]][p104]; and on setting, set this element's [[CryptographicNonce]][p104] to the given value.

¶[p104]

Note

Note how the setter for the `nonce`[p104] *IDL attribute does not update the corresponding content attribute. This, as well as the below setting of the* `nonce`[p104] *content attribute to the empty string when an element becomes browsing-context connected*[p47]*, 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 HTMLOrSVGOrMathMLElement, 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 HTMLOrSVGOrMathMLElement 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 policy container's CSP list.
2. If *CSP list* contains a header-delivered Content Security Policy, and *element* has a nonce content attribute whose value is not the empty string:
    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.*

The cloning steps for elements that include HTMLOrSVGOrMathMLElement given *node*, *copy*, and *subtree* are to set *copy*'s [[CryptographicNonce]] to *node*'s [[CryptographicNonce]].

2.5.7 Lazy loading attributes

A **lazy loading attribute** is an enumerated attribute with the following keywords and states:

Keyword	State	Brief description
`lazy`	**Lazy**	Used to defer fetching a resource until some conditions are met.
`eager`	**Eager**	Used to fetch a resource immediately; the default state.

The attribute directs the user agent to fetch a resource immediately or to defer fetching until some conditions associated with the element are met, according to the attribute's current state.

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, audio, video, and iframe element has associated **lazy load resumption steps**, initially null.

Each video element also has associated **poster lazy load resumption steps**, initially null.

Note

For `img`[p359], `audio`[p425], `video`[p420], and `iframe`[p404] elements that will lazy load[p105], these steps are run from the lazy load intersection observer[p106]'s callback or when their lazy loading attribute[p105] is set to the Eager[p105] state. This causes the element to continue loading. For `video`[p420] elements, poster lazy load resumption steps[p105] are also run at the same time.

Each `Document`[p135] 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[p106] 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#464 which tracks this. [INTERSECTIONOBSERVER][p1583]

   - 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[p105].
        3. Let *posterResumptionSteps* be null.
        4. If *entry*.`isIntersecting` is true and *entry*.`target` is a `video`[p420] element, then set *posterResumptionSteps* to the `video`[p420] element's poster lazy load resumption steps[p105].
        5. If *resumptionSteps* is null and *posterResumptionSteps* is null, then continue.
        6. Stop intersection-observing a lazy loading element[p107] for *entry*.`target`.
        7. If *resumptionSteps* is not null:
           1. Set *entry*.`target`'s lazy load resumption steps[p105] to null.
           2. Invoke *resumptionSteps*.
        8. If *posterResumptionSteps* is not null:
           1. Set the `video`[p420] element's poster lazy load resumption steps[p105] to null.
           2. Invoke *posterResumptionSteps*.

     The intention is to use the original value of the `isIntersecting` and `target` getters. See w3c/IntersectionObserver#464. [INTERSECTIONOBSERVER][p1583]

   - The *options* is an `IntersectionObserverInit` dictionary with the following dictionary members: «["`scrollMargin`" → lazy load scroll margin[p107]]»

     Note

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

     The lazy load scroll margin[p107] suggestions imply dynamic changes to the value, but the `IntersectionObserver` API does not support changing the scroll margin. See issue w3c/IntersectionObserver#428.

3. Call *doc*'s lazy load intersection observer[p106]'s `observe` method with *element* as the argument.

   The intention is to use the original value of the `observe` method. See w3c/IntersectionObserver#464.

[INTERSECTIONOBSERVER][p1583]

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[p106] is not null.
3. Call *doc*'s lazy load intersection observer[p106]'s `unobserve` method with *element* as the argument.

   The intention is to use the original value of the `unobserve` method. See w3c/IntersectionObserver#464. [INTERSECTIONOBSERVER][p1583]

The **lazy load scroll 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.

Note

It is important for privacy that the lazy load scroll margin[p107] 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.

2.5.8 Blocking attributes §[p107]

A **blocking attribute** explicitly indicates that certain operations should be blocked on the fetching of an external resource. The operations that can be blocked are represented by **possible blocking tokens**, which are strings listed by the following table:

Possible blocking token	Description
"`render`"	The element is potentially render-blocking[p107].

¶[p107] Note

In the future, there might be more possible blocking tokens[p107].

A blocking attribute[p107] must have a value that is an unordered set of unique space-separated tokens[p99], each of which are possible blocking tokens[p107]. The supported tokens of a blocking attribute[p107] are the possible blocking tokens[p107]. Any element can have at most one blocking attribute[p107].

The **blocking tokens set** for an element *el* are the result of the following steps:

1. Let *value* be the value of *el*'s blocking attribute[p107], or the empty string if no such attribute exists.
2. Set *value* to *value*, converted to ASCII lowercase.
3. Let *rawTokens* be the result of splitting *value* on ASCII whitespace.
4. Return a set containing the elements of *rawTokens* that are possible blocking tokens[p107].

An element is **potentially render-blocking** if its blocking tokens set[p107] contains "`render`[p107]", or if it is **implicitly potentially render-blocking**, which will be defined at the individual elements. By default, an element is not implicitly potentially render-blocking[p107].

2.5.9 Fetch priority attributes §[p108]

A **fetch priority attribute** is an enumerated attribute[p80] with the following keywords and states:

Keyword	State	Brief description
`high`	**High**	Signals a high-priority fetch relative to other resources with the same destination.
`low`	**Low**	Signals a low-priority fetch relative to other resources with the same destination.
`auto`	**Auto**	Signals automatic determination of fetch priority relative to other resources with the same destination.

The attribute's *missing value default*[p80] and *invalid value default*[p80] are both the Auto[p108] state.

2.6 Common DOM interfaces §[p108]

2.6.1 Reflecting content attributes in IDL attributes §[p108]

The building blocks for reflecting are as follows:

- A **reflected target** is an element or `ElementInternals`[p809] object. It is typically clear from context and typically identical to the interface of the reflected IDL attribute[p108]. It is always identical to that interface when it is an `ElementInternals`[p809] object.
- A **reflected IDL attribute** is an attribute interface member.
- A **reflected content attribute name** is a string. When the reflected target[p108] is an element, it represents the local name of a content attribute whose namespace is null. When the reflected target[p108] is an `ElementInternals`[p809] object, it represents a key of the reflected target[p108]'s target element[p810]'s internal content attribute map[p813].

A reflected IDL attribute[p108] can be defined to **reflect** a reflected content attribute name[p108] of a reflected target[p108]. In general this means that the IDL attribute getter returns the current value of the content attribute, and the setter changes the value of the content attribute to the given value.

Reflected targets[p108] have these associated algorithms:

- **get the element**: takes no arguments; returns an element.
- **get the content attribute**: takes no arguments; returns null or a string.
- **set the content attribute**: takes a string *value*; returns nothing.
- **delete the content attribute**: takes no arguments; returns nothing.

For a reflected target[p108] that is an element *element*, these are defined as follows:

get the element[p108]

1. Return *element*.

get the content attribute[p108]

1. Let *attribute* be the result of getting an attribute by namespace and local name given null, the reflected content attribute name[p108], and *element*.
2. If *attribute* is null, then return null.
3. Return *attribute*'s value.

set the content attribute[p108] **with a string *value***

1. Set an attribute value given *element*, the reflected content attribute name[p108], and *value*.

delete the content attribute[p108]

1. Remove an attribute by namespace and local name given null, the reflected content attribute name[p108], and *element*.

For a reflected target[p108] that is an `ElementInternals`[p809] object *elementInternals*, they are defined as follows:

get the element[p108]

1. Return *elementInternals*'s target element[p810].

get the content attribute[p108]

1. If *elementInternals*'s target element[p810]'s internal content attribute map[p813][the reflected content attribute name[p108]] does not exist, then return null.
2. Return *elementInternals*'s target element[p810]'s internal content attribute map[p813][the reflected content attribute name[p108]].

set the content attribute[p108] **with a string *value***

1. Set *elementInternals*'s target element[p810]'s internal content attribute map[p813][the reflected content attribute name[p108]] to *value*.

delete the content attribute[p108]

1. Remove *elementInternals*'s target element[p810]'s internal content attribute map[p813][the reflected content attribute name[p108]].

Note

This results in somewhat redundant data structures for `ElementInternals`[p809] objects as their target element[p810]'s internal content attribute map[p813] cannot be directly manipulated and as such reflection is only happening in a single direction. This approach was nevertheless chosen to make it less error-prone to define IDL attributes that are shared between reflected targets[p108] and benefit from common API semantics.

IDL attributes of type `DOMString` or `DOMString?` that reflect[p108] enumerated[p80] content attributes can be **limited to only known values**. Per the processing models below, those will cause the getters for such IDL attributes to only return keywords for those enumerated attributes, or the empty string or null.

If a reflected IDL attribute[p108] has the type `DOMString`:

- The getter steps are:
    1. Let *element* be the result of running this's get the element[p108].
    2. Let *contentAttributeValue* be the result of running this's get the content attribute[p108].
    3. Let *attributeDefinition* be the attribute definition of *element*'s content attribute whose namespace is null and local name is the reflected content attribute name[p108].
    4. If *attributeDefinition* indicates it is an enumerated attribute[p80] and the reflected IDL attribute[p108] is defined to be limited to only known values[p109]:
        1. If *contentAttributeValue* does not correspond to any state of *attributeDefinition* (e.g., it is null and there is no *missing value default*[p80]), or if it is in a state of *attributeDefinition* with no associated keyword value, then return the empty string.
        2. Return the canonical keyword[p80] for the state of *attributeDefinition* that *contentAttributeValue* corresponds to.
    5. If *contentAttributeValue* is null, then return the empty string.
    6. Return *contentAttributeValue*.
- The setter steps are to run this's set the content attribute[p108] with the given value.

If a reflected IDL attribute[p108] has the type `DOMString?`:

- The getter steps are:
    1. Let *element* be the result of running this's get the element[p108].
    2. Let *contentAttributeValue* be the result of running this's get the content attribute[p108].
    3. Let *attributeDefinition* be the attribute definition of *element*'s content attribute whose namespace is null and local name is the reflected content attribute name[p108].
    4. If *attributeDefinition* indicates it is an enumerated attribute[p80]:

1. Assert: the reflected IDL attribute[p108] is limited to only known values[p109].
2. Assert: *contentAttributeValue* corresponds to a state of *attributeDefinition*.
3. If *contentAttributeValue* corresponds to a state of *attributeDefinition* with no associated keyword value, then return null.
4. Return the canonical keyword[p80] for the state of *attributeDefinition* that *contentAttributeValue* corresponds to.

5. Return *contentAttributeValue*.

- The setter steps are:

1. If the given value is null, then run this's delete the content attribute[p108].
2. Otherwise, run this's set the content attribute[p108] with the given value.

If a reflected IDL attribute[p108] has the type `USVString`, optionally **treated as a URL**:

- The getter steps are:

1. Let *element* be the result of running this's get the element[p108].
2. Let *contentAttributeValue* be the result of running this's get the content attribute[p108].
3. If the reflected IDL attribute[p108] is treated as a URL[p110]:
    1. If *contentAttributeValue* is null, then return the empty string.
    2. Let *urlString* be the result of encoding-parsing-and-serializing a URL[p101] given *contentAttributeValue*, relative to *element*'s node document.
    3. If *urlString* is not failure, then return *urlString*.
4. Return *contentAttributeValue*, converted to a scalar value string.

- The setter steps are to run this's set the content attribute[p108] with the given value.

If a reflected IDL attribute[p108] has the type `boolean`:

- The getter steps are:

1. Let *contentAttributeValue* be the result of running this's get the content attribute[p108].
2. If *contentAttributeValue* is null, then return false.
3. Return true.

- The setter steps are:

1. If the given value is false, then run this's delete the content attribute[p108].
2. If the given value is true, then run this's set the content attribute[p108] with the empty string.

Note

This corresponds to the rules for boolean content attributes[p79].

If a reflected IDL attribute[p108] has the type `long`, optionally **limited to only non-negative numbers** and optionally with a **default value** *defaultValue*:

- The getter steps are:

1. Let *contentAttributeValue* be the result of running this's get the content attribute[p108].
2. If *contentAttributeValue* is not null:
    1. Let *parsedValue* be the result of integer parsing[p80] *contentAttributeValue* if the reflected IDL attribute[p108]

is not limited to only non-negative numbers[p110]; otherwise the result of non-negative integer parsing[p81] *contentAttributeValue*.

2. If *parsedValue* is not an error and is within the `long` range, then return *parsedValue*.

3. If the reflected IDL attribute[p108] has a default value[p110], then return *defaultValue*.

4. If the reflected IDL attribute[p108] is limited to only non-negative numbers[p110], then return −1.

5. Return 0.

- The setter steps are:

1. If the reflected IDL attribute[p108] is limited to only non-negative numbers[p110] and the given value is negative, then throw an `"IndexSizeError"` `DOMException`.

2. Run this's set the content attribute[p108] with the given value converted to the shortest possible string representing the number as a valid integer[p80].

If a reflected IDL attribute[p108] has the type `unsigned long`, optionally **limited to only positive numbers**, **limited to only positive numbers with fallback**, or **clamped to the range** [*clampedMin*, *clampedMax*], and optionally with a default value[p110] *defaultValue*:

- The getter steps are:

1. Let *contentAttributeValue* be the result of running this's get the content attribute[p108].

2. Let *minimum* be 0.

3. If the reflected IDL attribute[p108] is limited to only positive numbers[p111] or limited to only positive numbers with fallback[p111], then set *minimum* to 1.

4. If the reflected IDL attribute[p108] is clamped to the range[p111], then set *minimum* to *clampedMin*.

5. Let *maximum* be 2147483647 if the reflected IDL attribute[p108] is not clamped to the range[p111]; otherwise *clampedMax*.

6. If *contentAttributeValue* is not null:

1. Let *parsedValue* be the result of non-negative integer parsing[p81] *contentAttributeValue*.

2. If *parsedValue* is not an error and is in the range *minimum* to *maximum*, inclusive, then return *parsedValue*.

3. If *parsedValue* is not an error and the reflected IDL attribute[p108] is clamped to the range[p111]:

1. If *parsedValue* is less than *minimum*, then return *minimum*.

2. Return *maximum*.

7. If the reflected IDL attribute[p108] has a default value[p110], then return *defaultValue*.

8. Return *minimum*.

- The setter steps are:

1. If the reflected IDL attribute[p108] is limited to only positive numbers[p111] and the given value is 0, then throw an `"IndexSizeError"` `DOMException`.

2. Let *minimum* be 0.

3. If the reflected IDL attribute[p108] is limited to only positive numbers[p111] or limited to only positive numbers with fallback[p111], then set *minimum* to 1.

4. Let *newValue* be *minimum*.

5. If the reflected IDL attribute[p108] has a default value[p110], then set *newValue* to *defaultValue*.

6. If the given value is in the range *minimum* to 2147483647, inclusive, then set *newValue* to it.

7. Run this's set the content attribute[p108] with *newValue* converted to the shortest possible string representing the

number as a valid non-negative integer[p81].

Note

Clamped to the range[p111] has no effect on the setter steps.

If a reflected IDL attribute[p108] has the type `double`, optionally limited to only positive numbers[p111] and optionally with a default value[p110] *defaultValue*:

- The getter steps are:

  1. Let *contentAttributeValue* be the result of running this's get the content attribute[p108].
  2. If *contentAttributeValue* is not null:
     1. Let *parsedValue* be the result of floating-point number parsing[p82] *contentAttributeValue*.
     2. If *parsedValue* is not an error and is greater than 0, then return *parsedValue*.
     3. If *parsedValue* is not an error and the reflected IDL attribute[p108] is not limited to only positive numbers[p111], then return *parsedValue*.
  3. If the reflected IDL attribute[p108] has a default value[p110], then return *defaultValue*.
  4. Return 0.

- The setter steps are:

  1. If the reflected IDL attribute[p108] is limited to only positive numbers[p111] and the given value is not greater than 0, then return.
  2. Run this's set the content attribute[p108] with the given value, converted to the best representation of the number as a floating-point number[p82].

Note

The values Infinity and Not-a-Number (NaN) values throw an exception on setting, as defined in Web IDL. *[WEBIDL][p1588]*

If a reflected IDL attribute[p108] has the type `DOMTokenList`, then its getter steps are to return a `DOMTokenList` object whose associated element is this and associated attribute's local name is the reflected content attribute name[p108]. Specification authors cannot reflect IDL attributes of this type on `ElementInternals`[p809].

If a reflected IDL attribute[p108] has the type *T*`?`, where *T* is either `Element` or an interface that inherits from `Element`, then with *attr* being the reflected content attribute name[p108]:

- Its reflected target[p108] has an **explicitly set *attr*-element**, which is a weak reference to an element or null. It is initially null.
- Its reflected target[p108] *reflectedTarget* has a **get the *attr*-associated element** algorithm, that runs these steps:

  1. Let *element* be the result of running *reflectedTarget*'s get the element[p108].
  2. Let *contentAttributeValue* be the result of running *reflectedTarget*'s get the content attribute[p108].
  3. If *reflectedTarget*'s explicitly set *attr*-element[p112] is not null:
     1. If *reflectedTarget*'s explicitly set *attr*-element[p112] is a descendant of any of *element*'s shadow-including ancestors, then return *reflectedTarget*'s explicitly set *attr*-element[p112].
     2. Return null.
  4. Otherwise, if *contentAttributeValue* is not null, return the first element *candidate*, in tree order, that meets the following criteria:
     - *candidate*'s root is the same as *element*'s root;
     - *candidate*'s ID is *contentAttributeValue*; and
     - *candidate* implements *T*.

If no such element exists, then return null.

5. Return null.

- The getter steps are to return the result of running this's get the *attr*-associated element[p112].
- The setter steps are:

    1. If the given value is null:

        1. Set this's explicitly set *attr*-element[p112] to null.
        2. Run this's delete the content attribute[p108].
        3. Return.

    2. Run this's set the content attribute[p108] with the empty string.
    3. Set this's explicitly set *attr*-element[p112] to a weak reference to the given value.

- For element reflected targets[p108] only: the following attribute change steps, given *element*, *localName*, *oldValue*, *value*, and *namespace*, are used to synchronize between the content attribute and the IDL attribute:

    1. If *localName* is not *attr* or *namespace* is not null, then return.
    2. Set *element*'s explicitly set *attr*-element[p112] to null.

Note

Reflected IDL attributes[p108] of this type are strongly encouraged to have their identifier end in "Element" for consistency.

If a reflected IDL attribute[p108] has the type `FrozenArray<T>?`, where *T* is either `Element` or an interface that inherits from `Element`, then with *attr* being the reflected content attribute name[p108]:

- Its reflected target[p108] has an **explicitly set *attr*-elements**, which is either a list of weak references to elements or null. It is initially null.
- Its reflected target[p108] has a **cached *attr*-associated elements**, which is a list of elements. It is initially « ».
- Its reflected target[p108] has a **cached *attr*-associated elements object**, which is a `FrozenArray<T>?`. It is initially null.
- Its reflected target[p108] *reflectedTarget* has a **get the *attr*-associated elements** algorithm, which runs these steps:

    1. Let *elements* be an empty list.
    2. Let *element* be the result of running *reflectedTarget*'s get the element[p108].
    3. If *reflectedTarget*'s explicitly set *attr*-elements[p113] is not null:

        1. For each *attrElement* in *reflectedTarget*'s explicitly set *attr*-elements[p113]:

            1. If *attrElement* is not a descendant of any of *element*'s shadow-including ancestors, then continue.
            2. Append *attrElement* to *elements*.

    4. Otherwise:

        1. Let *contentAttributeValue* be the result of running *reflectedTarget*'s get the content attribute[p108].
        2. If *contentAttributeValue* is null, then return null.
        3. Let *tokens* be *contentAttributeValue*, split on ASCII whitespace.
        4. For each *id* of *tokens*:

            1. Let *candidate* be the first element, in tree order, that meets the following criteria:

                - *candidate*'s root is the same as *element*'s root;
                - *candidate*'s ID is *id*; and

- *candidate* implements *T*.

If no such element exists, then continue.

2. Append *candidate* to *elements*.

5. Return *elements*.

- The getter steps are:

1. Let *elements* be the result of running this's get the *attr*-associated elements[p113].
2. If the contents of *elements* is equal to the contents of this's cached *attr*-associated elements[p113], then return this's cached *attr*-associated elements object[p113].
3. Let *elementsAsFrozenArray* be *elements*, converted to a `FrozenArray<T>?`.
4. Set this's cached *attr*-associated elements[p113] to *elements*.
5. Set this's cached *attr*-associated elements object[p113] to *elementsAsFrozenArray*.
6. Return *elementsAsFrozenArray*.

Note

This extra caching layer is necessary to preserve the invariant that `element.reflectedElements === element.reflectedElements`.

- The setter steps are:

1. If the given value is null:
    1. Set this's explicitly set *attr*-elements[p113] to null.
    2. Run this's delete the content attribute[p108].
    3. Return.
2. Run this's set the content attribute[p108] with the empty string.
3. Let *elements* be an empty list.
4. For each *element* in the given value:
    1. Append a weak reference to *element* to *elements*.
5. Set this's explicitly set *attr*-elements[p113] to *elements*.

- For element reflected targets[p108] only: the following attribute change steps, given *element*, *localName*, *oldValue*, *value*, and *namespace*, are used to synchronize between the content attribute and the IDL attribute:

1. If *localName* is not *attr* or *namespace* is not null, then return.
2. Set *element*'s explicitly set *attr*-elements[p113] to null.

Note

Reflected IDL attributes[p108] of this type are strongly encouraged to have their identifier end in `"Elements"` *for consistency.*

2.6.2 Using reflect via IDL extended attributes §[p114]

Reflection[p108] can be used from IDL through extended attributes. **`[Reflect]`**, **`[ReflectSetter]`**, **`[ReflectURL]`**, **`[ReflectNonNegative]`**, **`[ReflectPositive]`**, and **`[ReflectPositiveWithFallback]`** all trigger reflection[p108]. These must either take no arguments, or take a string; they must not appear on anything other than an interface member attribute; and only one of these can be used at a time.

For one of these primary reflection extended attributes, its reflected content attribute name[p108] is the string value it takes, if one is

provided; otherwise it is the IDL attribute name converted to ASCII lowercase.

IDL attributes with the [Reflect] extended attribute must reflect [Reflect]'s reflected content attribute name.

IDL attributes with the [ReflectSetter] extended attribute on setting must reflect [ReflectSetter]'s reflected content attribute name.

The [ReflectURL] extended attribute must only appear on attributes with a type of USVString.

IDL attributes with the [ReflectURL] extended attribute must reflect, as a URL, [ReflectURL]'s reflected content attribute name.

The [ReflectNonNegative] extended attribute must only appear on attributes with a type of long.

IDL attributes with the [ReflectNonNegative] extended attribute must reflect, limited to only non-negative numbers, [ReflectNonNegative]'s reflected content attribute name.

The [ReflectPositive] and [ReflectPositiveWithFallback] extended attributes must only appear on attributes with a type of double or unsigned long.

IDL attributes with the [ReflectPositive] extended attribute must reflect, limited to only positive numbers, [ReflectPositive]'s reflected content attribute name.

IDL attributes with the [ReflectPositiveWithFallback] extended attribute must reflect, limited to only positive numbers with fallback, [ReflectPositiveWithFallback]'s reflected content attribute name.

To supplement the above extended attributes we also introduce **[ReflectRange]**, and **[ReflectDefault]**. These augment how reflection works and also must only appear on interface member attributes.

The [ReflectRange] extended attribute must take an integer list limited to two values. It must only be used on attributes with a type of unsigned long. Additionally, it must also only appear alongside [Reflect].

IDL attributes with the [ReflectRange] extended attribute are clamped to the range [*clampedMin*, *clampedMax*] where *clampedMin* is the first, and *clampedMax* is the second argument to the list provided to [ReflectRange].

The [ReflectDefault] extended attribute must only be used on attributes with a type of double, long, or unsigned long. When used on an attribute of type double, it must take a decimal; otherwise it must take an integer. Additionally, it must also only appear alongside [Reflect], [ReflectNonNegative], [ReflectPositive], or [ReflectPositiveWithFallback].

IDL attributes with the [ReflectDefault] extended attribute have a default value provided by the argument provided to [ReflectDefault].

2.6.3 Using reflect in specifications §

Reflection is primarily about improving web developer ergonomics by giving them typed access to content attributes through reflected IDL attributes. The ultimate source of truth, which the web platform builds upon, is the content attributes themselves. That is, specification authors must not use the reflected IDL attribute getter or setter steps, but instead must use the content attribute presence and value. (Or an abstraction on top, such as the state of an enumerated attribute.)

Two important exceptions to this are reflected IDL attributes whose type is one of the following:

- *T*?, where *T* is either Element or an interface that inherits from Element
- FrozenArray<*T*>?, where *T* is either Element or an interface that inherits from Element

For those, specification authors must use the reflected target's get the *attr*-associated element and get the *attr*-associated elements, respectively. The content attribute presence and value must not be used as they cannot be fully synchronized with the reflected IDL attribute.

A reflected target's explicitly set *attr*-element, explicitly set *attr*-elements, cached *attr*-associated elements, and cached *attr*-associated elements object are to be treated as internal implementation details and not to be built upon.

2.6.4 Collections §p116

The HTMLFormControlsCollection[p117] and HTMLOptionsCollection[p119] interfaces are collections derived from the HTMLCollection interface. The HTMLAllCollection[p116] interface is a collection, but is not so derived.

2.6.4.1 The HTMLAllCollection[p116] interface §p116

The HTMLAllCollection[p116] interface is used for the legacy document.all[p1544] 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.

Note

All HTMLAllCollection[p116] objects are rooted at a Document[p135] and have a filter that matches all elements, so the elements represented by the collection of an HTMLAllCollection[p116] object consist of all the descendant elements of the root Document[p135].

Objects that implement the HTMLAllCollection[p116] interface are legacy platform objects with an additional [[Call]] internal method described in the section below[p117]. They also have an [[IsHTMLDDA]] internal slot.

Note

Objects that implement the HTMLAllCollection[p116] interface have several unusual behaviors, due of the fact that they have an [[IsHTMLDDA]] internal slot:

- *The ToBoolean abstract operation in JavaScript returns false when given objects implementing the HTMLAllCollection[p116] interface.*
- *The IsLooselyEqual abstract operation, when given objects implementing the HTMLAllCollection[p116] interface, returns true when compared to the `undefined` and `null` values. (Comparisons using the IsStrictlyEqual abstract operation, and IsLooselyEqual 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[p116] 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[p1544] as a way to detect legacy user agents, and one that only supports those legacy user agents and uses the document.all[p1544] object without testing for its presence first. [JAVASCRIPT][p1583]

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.
};
```

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[p162] 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[p117] represented by the collection, in tree order, ignoring later duplicates, with the id[p162] 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 **`length`** getter steps are to return the number of nodes represented by the collection.

The indexed property getter must return the result of getting the "all"-indexed element[p117] from this given the passed index.

The **`namedItem(`*`name`*`)`** method steps are to return the result of getting the "all"-named element(s)[p117] from this given *name*.

The **item(*nameOrIndex*)** method steps are:

1. If *nameOrIndex* was not provided, return null.
2. Return the result of getting the "all"-indexed or named element(s)[p117] from this, given *nameOrIndex*.

The following elements are **"all"-named elements**: a[p267], button[p585], embed[p413], form[p532], frame[p1537], frameset[p1537], iframe[p404], img[p359], input[p539], map[p487], meta[p196], object[p416], select[p590], and textarea[p606]

To **get the "all"-indexed element** from an HTMLAllCollection[p116] *collection* given an index *index*, return the *index*th element in *collection*, or null if there is no such *index*th element.

To **get the "all"-named element(s)** from an HTMLAllCollection[p116] *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[p135] as *collection*, whose filter matches only elements that are either:
   - "all"-named elements[p117] 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[p116] *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[p117] from *collection* given the number represented by *nameOrIndex*.
2. Return the result of getting the "all"-named element(s)[p117] from *collection* given *nameOrIndex*.

2.6.4.1.1 [[Call]] (thisArgument, argumentsList) §[p117]

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)[p117] from this HTMLAllCollection[p116] given *nameOrIndex*.
4. Return the result of converting *result* to an ECMAScript value.

Note

The thisArgument *is ignored, and thus code such as* `Function.prototype.call.call(document.all, null, "x")` *will still search for elements. (*`document.all.call` *does not exist, since* `document.all` *does not inherit from* `Function.prototype`*.)*

2.6.4.2 The HTMLFormControlsCollection[p117] interface §[p117]

The HTMLFormControlsCollection[p117] interface is used for collections of listed elements[p532] in form[p532] elements.

IDL

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

```
[Exposed=Window]
interface RadioNodeList : NodeList {
  attribute DOMString value;
};
```

For web developers (non-normative)

***collection*.length**

Returns the number of elements in *collection*.

***element* = *collection*.item(*index*)**

***element* = *collection*[*index*]**

Returns the item at index *index* in *collection*. The items are sorted in tree order.

***element* = *collection*.namedItem(*name*)**

***radioNodeList* = *collection*.namedItem(*name*)**

***element* = *collection*[*name*]**

***radioNodeList* = *collection*[*name*]**

Returns the item with ID or name *name* from *collection*.

If there are multiple matching items, then a RadioNodeList object containing all those elements is returned.

***radioNodeList*.value**

Returns the value of the first checked radio button represented by *radioNodeList*.

radioNodeList*.value = *value

Checks the first radio button represented by *radioNodeList* that has value *value*.

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[p118] object that is an input[p539] element whose type[p541] attribute is in the Radio Button[p561] state and whose value[p543] 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[p118] object that is an input[p539] element whose type[p541] attribute is in the Radio Button[p561] state and whose value[p543] 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[p626] to true.

2.6.4.3 The HTMLOptionsCollection[p119] interface §[p119]

The HTMLOptionsCollection[p119] interface is used for collections of option[p601] elements. It is always rooted on a select[p590] element and has attributes and methods that manipulate that element's descendants.

IDL

```
[Exposed=Window]
interface HTMLOptionsCollection : HTMLCollection {
  // inherits 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);
  [CEReactions] attribute long selectedIndex;
};
```

For web developers (non-normative)

***collection*.length[p120]**

Returns the number of elements in *collection*.

collection*.length[p120] = *value

When set to a smaller number than the existing length, truncates the number of option[p601] elements in the container corresponding to *collection*.

When set to a greater number than the existing length, if that number is less than or equal to 100000, adds new blank option[p601] elements to the container corresponding to *collection*.

***element* = *collection*.item(*index*)**
***element* = *collection*[*index*]**

Returns the item at index *index* in *collection*. The items are sorted in tree order.

collection*[*index*] = *element

When *index* is a greater number than the number of items in *collection*, adds new blank option[p601] elements in the corresponding container.

When set to null, removes the item at index *index* from *collection*.

When set to an option[p601] element, adds or replaces it at index *index* in *collection*.

***element* = *collection*.namedItem(*name*)**
***element* = *collection*[*name*]**

Returns the item with ID or name[p1531] *name* from *collection*.

If there are multiple matching items, then the first is returned.

***collection*.add[p120](*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 *collection*, in which case *element* is inserted before that element.

If *before* is omitted, null, or a number out of range, then *element* will be added at the end of the list.

Throws a "HierarchyRequestError" DOMException if *element* is an ancestor of the element into which it is to be inserted.

***collection*.remove(*index*)**

Removes the item with index *index* from *collection*.

***collection*.selectedIndex**

Returns the index of the first selected item, if any, or −1 if there is no selected item.

collection*.selectedIndex = *index

Changes the selection to the option element at index *index* in *collection*.

The object's supported property indices are as defined for HTMLCollection objects.

The **length** getter steps are to return the number of nodes represented by the collection.

The length setter steps are:

1. Let *current* be the number of nodes represented by the collection.
2. If the given value is greater than *current*:
    1. If the given value is greater than 100,000, then return.
    2. Let *n* be *value* − *current*.
    3. Append new option elements to the select element on which this is rooted given *n*.
3. If the given value is less than *current*:
    1. Let *n* be *current* − *value*.
    2. Remove the last *n* nodes in the collection from their parent nodes.

Note

Setting length never removes or adds any optgroup elements, and never adds new children to existing optgroup elements (though it can remove children from them).

The supported property names consist of the non-empty values of all the id and name attributes of all the elements represented by the collection, in tree order, ignoring later duplicates, with the id of an element preceding its name if it contributes both, they differ from each other, and neither is the duplicate of an earlier entry.

To **append new option elements** to a select element *select* given a non-negative integer *count*:

1. Let *fragment* be the result of creating a document fragment given *select*'s node document.
2. Append *count* new option elements to *fragment*.
3. Append *fragment* to *select*.

To set the value of a new indexed property or set the value of an existing indexed property for an HTMLOptionsCollection *collection*, given a property index *index* and a new value *value*:

1. If *value* is null, then remove an option from *collection* given *index* and return.
2. Let *length* be the number of nodes represented by *collection*.
3. Let *delta* be *index* − *length*.
4. If *delta* is greater than 0, then append new option elements to the select element on which *collection* is rooted given *delta*.
5. If *delta* is greater than or equal to 0, then append *value* to the select element on which *collection* is rooted. Otherwise, replace the *index*th element in *collection* by *value*.

The **add(*element*, *before*)** method steps are:

1. If *element* is an ancestor of the select element on which this is rooted, then throw a "HierarchyRequestError"

DOMException.

2. If *before* is an element, but that element isn't a descendant of the select[p590] element on which this is rooted, then throw a "NotFoundError" DOMException.
3. If *element* and *before* are the same element, then return.
4. Let *reference* be null.
5. If *before* is a node, then set *reference* to *before*. Otherwise, if *before* is an integer and there is a *before*th node in this, then set *reference* to that node.
6. Let *parent* be *reference*'s parent node if *reference* is not null; otherwise the select[p590] element on which this is rooted.
7. Pre-insert *element* into *parent* node before *reference*.

To **remove an option** from an HTMLOptionsCollection[p119] *collection* given an integer *index*:

1. If the number of nodes represented by *collection* is 0, then return.
2. If *index* is not a number greater than or equal to 0 and less than the number of nodes represented by *collection*, then return.
3. Let *element* be the *index*th element in *collection*.
4. Remove *element* from its parent node.

The **remove(*index*)** method steps are to remove an option[p121] from this given *index*.

The **selectedIndex** getter steps are to return the selected index[p595] of the select[p590] element on which this is rooted.

The selectedIndex[p121] setter steps are to set the selected index[p595] of the select[p590] element on which this is rooted to the given value.

2.6.5 The DOMStringList[p121] interface §[p12 1]

The DOMStringList[p121] 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[p121]***.***

For web developers (non-normative)

***strings*.length**[p122]

Returns the number of strings in *strings*.

***strings*[*index*]**

***strings*.item**[p122]**(*index*)**

Returns the string with index *index* from *strings*.

***strings*.contains**[p122]**(*string*)**

Returns true if *strings* contains *string*, and false otherwise.

Each DOMStringList[p121] object has an associated list.

The DOMStringList[p121] interface supports indexed properties. The supported property indices are the indices of this's associated list.

The **length** getter steps are to return this's associated list's size.

The **item(*index*)** method steps are to return the *index*th item in this's associated list, or null if *index* plus one is greater than this's associated list's size.

The **contains(*string*)** method steps are to return true if this's associated list contains *string*, and false otherwise.

2.7 Safe passing of structured data §[p12]

To support passing JavaScript objects, including platform objects, across realm boundaries, this specification defines the following infrastructure for serializing and deserializing objects, including in some cases transferring the underlying data instead of copying it. Collectively this serialization/deserialization process is known as "structured cloning", although most APIs perform separate serialization and deserialization steps. (With the notable exception being the `structuredClone()`[p134] method.)

This section uses the terminology and typographic conventions from the JavaScript specification. [JAVASCRIPT][p1583]

2.7.1 Serializable objects §[p12]

Serializable objects[p122] support being serialized, and later deserialized, in a way that is independent of any given 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[p122], and not all aspects of objects that are serializable objects[p122] are necessarily preserved when they are serialized.

Platform objects can be serializable objects[p122] 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 realm.

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

These steps may perform a sub-serialization[p127] to serialize nested data structures. They should not call StructuredSerialize[p128] 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*, a platform object *value*, and a realm *targetRealm

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[p130] to deserialize nested data structures. They should not call StructuredDeserialize[p128] 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]`[p122] 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]`[p122]-annotated interface in the inheritance chain needs to define standalone serialization steps[p122] and deserialization steps[p122], 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 be serializable objects[p122] by annotating the `Person` interface with the `[Serializable]`[p122] extended attribute, and defining the following accompanying algorithms:

Their serialization steps[p122], given *value* and *serialized*:

1. Set *serialized*.[[Name]] to *value*'s associated name value.
2. Let *serializedBestFriend* be the sub-serialization[p127] of *value*'s associated best friend value.
3. Set *serialized*.[[BestFriend]] to *serializedBestFriend*.

Their deserialization steps[p122], given *serialized*, *value*, and *targetRealm*:

1. Set *value*'s associated name value to *serialized*.[[Name]].
2. Let *deserializedBestFriend* be the sub-deserialization[p130] of *serialized*.[[BestFriend]].
3. Set *value*'s associated best friend value to *deserializedBestFriend*.

Objects defined in the JavaScript specification are handled by the StructuredSerialize[p128] abstract operation directly.

¶ p123

Note

Originally, this specification defined the concept of "cloneable objects", which could be cloned from one 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 § p123

Transferable objects[p123] 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[p123] and not all aspects of objects that are transferable objects[p123] 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[p123] 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 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]p123 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]p123-annotated interface in the inheritance chain needs to define standalone transfer stepsp123 and transfer-receiving stepsp123, including taking into account any important data that might come from inherited interfaces.

Platform objects that are transferable objectsp123 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 StructuredSerializeWithTransferp131 abstract operation directly.

2.7.3 StructuredSerializeInternal (*value*, *forStorage* [, *memory*]) §p12 4

The StructuredSerializeInternalp124 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 *value* is undefined, null, a Boolean, a Number, a BigInt, or a String, then return { [[Type]]: "primitive", [[Value]]: *value* }.
5. If *value* is a 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:
    1. If IsSharedArrayBuffer(*value*) is true:
       1. If the current settings objectp1151's cross-origin isolated capabilityp1144 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[p1144] cannot change over time and a `SharedArrayBuffer` cannot leave an agent cluster.

2. If *forStorage* is true, then throw a "`DataCloneError`" `DOMException`.
3. If *value* has an [[ArrayBufferMaxByteLength]] internal slot, then set *serialized* to { [[Type]]: "GrowableSharedArrayBuffer", [[ArrayBufferData]]: *value*.[[ArrayBufferData]], [[ArrayBufferByteLengthData]]: *value*.[[ArrayBufferByteLengthData]], [[ArrayBufferMaxByteLength]]: *value*.[[ArrayBufferMaxByteLength]], [[AgentCluster]]: the surrounding agent's agent cluster }.
4. Otherwise, set *serialized* to { [[Type]]: "SharedArrayBuffer", [[ArrayBufferData]]: *value*.[[ArrayBufferData]], [[ArrayBufferByteLength]]: *value*.[[ArrayBufferByteLength]], [[AgentCluster]]: the surrounding agent's agent cluster }.

2. Otherwise:
    1. If IsDetachedBuffer(*value*) is true, then throw a "`DataCloneError`" `DOMException`.
    2. Let *size* be *value*.[[ArrayBufferByteLength]].
    3. Let *dataCopy* be ? CreateByteDataBlock(*size*).

        Note

        This can throw a `RangeError` exception upon allocation failure.

    4. Perform CopyDataBlockBytes(*dataCopy*, 0, *value*.[[ArrayBufferData]], 0, *size*).
    5. If *value* has an [[ArrayBufferMaxByteLength]] internal slot, then set *serialized* to { [[Type]]: "ResizableArrayBuffer", [[ArrayBufferData]]: *dataCopy*, [[ArrayBufferByteLength]]: *size*, [[ArrayBufferMaxByteLength]]: *value*.[[ArrayBufferMaxByteLength]] }.
    6. Otherwise, set *serialized* to { [[Type]]: "ArrayBuffer", [[ArrayBufferData]]: *dataCopy*, [[ArrayBufferByteLength]]: *size* }.

14. Otherwise, if *value* has a [[ViewedArrayBuffer]] internal slot:
    1. If IsArrayBufferViewOutOfBounds(*value*) is true, then throw a "`DataCloneError`" `DOMException`.
    2. Let *buffer* be the value of *value*'s [[ViewedArrayBuffer]] internal slot.
    3. Let *bufferSerialized* be ? StructuredSerializeInternal[p124](*buffer*, *forStorage*, *memory*).
    4. Assert: *bufferSerialized*.[[Type]] is "ArrayBuffer", "ResizableArrayBuffer", "SharedArrayBuffer", or "GrowableSharedArrayBuffer".
    5. If *value* has a [[DataView]] internal slot, then set *serialized* to { [[Type]]: "ArrayBufferView", [[Constructor]]: "DataView", [[ArrayBufferSerialized]]: *bufferSerialized*, [[ByteLength]]: *value*.[[ByteLength]], [[ByteOffset]]: *value*.[[ByteOffset]] }.
    6. 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:
    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:
    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:

    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. Let *stack* be an implementation-defined string that represents *value*.[[Stack]]. [JSERRORSTACKACCESSOR]p1583 [JSERRORSTACKS]p1583
    6. Set *serialized* to { [[Type]]: "Error", [[Name]]: *name*, [[Message]]: *message*, [[Stack]]: *stack* }.
    7. User agents should attach a serialized representation of any interesting accompanying data which are not yet specified to *serialized*.

18. Otherwise, if *value* is an Array exotic object:

    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 objectp122:

    1. If *value* has a [[Detached]]p124 internal slot whose value is true, then throw a "DataCloneError" DOMException.
    2. Let *typeString* 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]], [[Extensible]], or [[PrivateElements]], then throw a "DataCloneError" DOMException.

    Example

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

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

    Example

    For instance, a proxy object.

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:

1. If *value* has a [[MapData]] internal slot:
    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]]: *entry*.[[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[p124](*entry*.[[Key]], *forStorage*, *memory*).
        2. Let *serializedValue* be ? StructuredSerializeInternal[p124](*entry*.[[Value]], *forStorage*, *memory*).
        3. Append { [[Key]]: *serializedKey*, [[Value]]: *serializedValue* } to *serialized*.[[MapData]].
2. Otherwise, if *value* has a [[SetData]] internal slot:
    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[p124](*entry*, *forStorage*, *memory*).
        2. Append *serializedEntry* to *serialized*.[[SetData]].
3. Otherwise, if *value* is a platform object that is a serializable object[p122], then perform the serialization steps[p122] for *value*'s primary interface, given *value*, *serialized*, and *forStorage*.

    The serialization steps[p122] may need to perform a **sub-serialization**. This is an operation which takes as input a value *subValue*, and returns StructuredSerializeInternal[p124](*subValue*, *forStorage*, *memory*). (In other words, a sub-serialization[p127] is a specialization of StructuredSerializeInternal[p124] to be consistent within this invocation.)
4. Otherwise, for each *key* in ! EnumerableOwnProperties(*value*, key):
    1. If ! HasOwnProperty(*value*, *key*) is true:
        1. Let *inputValue* be ? *value*.[[Get]](*key*, *value*).
        2. Let *outputValue* be ? StructuredSerializeInternal[p124](*inputValue*, *forStorage*, *memory*).
        3. Append { [[Key]]: *key*, [[Value]]: *outputValue* } to *serialized*.[[Properties]].

27. Return *serialized*.

Example

It's important to realize that the Records produced by StructuredSerializeInternal[p124] might contain "pointers" to other records that create circular references. For example, when we pass the following JavaScript object into StructuredSerializeInternal[p124]:

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

it produces the following result:

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

2.7.4 StructuredSerialize (*value*) §p12 8

1. Return ? StructuredSerializeInternal[p124](*value*, false).

2.7.5 StructuredSerializeForStorage (*value*) §p12 8

1. Return ? StructuredSerializeInternal[p124](*value*, true).

2.7.6 StructuredDeserialize (*serialized*, *targetRealm* [, *memory*]) §p12 8

The StructuredDeserialize[p128] abstract operation takes as input a Record *serialized*, which was previously produced by StructuredSerialize[p128] or StructuredSerializeForStorage[p128], 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 *memory* was not supplied, let *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 *memory*[*serialized*] exists, then return *memory*[*serialized*].
3. Let *deep* be false.
4. Let *value* be an uninitialized value.
5. If *serialized*.[[Type]] is "primitive", then set *value* to *serialized*.[[Value]].
6. Otherwise, if *serialized*.[[Type]] is "Boolean", then set *value* to a new Boolean object in *targetRealm* whose [[BooleanData]] internal slot value is *serialized*.[[BooleanData]].
7. Otherwise, if *serialized*.[[Type]] is "Number", then set *value* to a new Number object in *targetRealm* whose [[NumberData]] internal slot value is *serialized*.[[NumberData]].
8. Otherwise, if *serialized*.[[Type]] is "BigInt", then set *value* to a new BigInt object in *targetRealm* whose [[BigIntData]] internal slot value is *serialized*.[[BigIntData]].
9. Otherwise, if *serialized*.[[Type]] is "String", then set *value* to a new String object in *targetRealm* whose [[StringData]] internal slot value is *serialized*.[[StringData]].
10. Otherwise, if *serialized*.[[Type]] is "Date", then set *value* to a new Date object in *targetRealm* whose [[DateValue]] internal slot value is *serialized*.[[DateValue]].
11. Otherwise, if *serialized*.[[Type]] is "RegExp", then set *value* to a new RegExp object in *targetRealm* whose [[RegExpMatcher]] internal slot value is *serialized*.[[RegExpMatcher]], whose [[OriginalSource]] internal slot value is *serialized*.[[OriginalSource]], and whose [[OriginalFlags]] internal slot value is *serialized*.[[OriginalFlags]].
12. Otherwise, if *serialized*.[[Type]] is "SharedArrayBuffer":
    1. If *targetRealm*'s corresponding agent cluster is not *serialized*.[[AgentCluster]], then throw a `"DataCloneError"` `DOMException`.
    2. Otherwise, set *value* to a new SharedArrayBuffer object in *targetRealm* whose [[ArrayBufferData]] internal slot value is *serialized*.[[ArrayBufferData]] and whose [[ArrayBufferByteLength]] internal slot value is *serialized*.[[ArrayBufferByteLength]].
13. Otherwise, if *serialized*.[[Type]] is "GrowableSharedArrayBuffer":
    1. If *targetRealm*'s corresponding agent cluster is not *serialized*.[[AgentCluster]], then throw a `"DataCloneError"` `DOMException`.
    2. Otherwise, set *value* to a new SharedArrayBuffer object in *targetRealm* whose [[ArrayBufferData]] internal slot

value is *serialized*.[[ArrayBufferData]], whose [[ArrayBufferByteLengthData]] internal slot value is *serialized*.[[ArrayBufferByteLengthData]], and whose [[ArrayBufferMaxByteLength]] internal slot value is *serialized*.[[ArrayBufferMaxByteLength]].

14. Otherwise, if *serialized*.[[Type]] is "ArrayBuffer", then set *value* to a new ArrayBuffer object in *targetRealm* whose [[ArrayBufferData]] internal slot value is *serialized*.[[ArrayBufferData]], and whose [[ArrayBufferByteLength]] internal slot value is *serialized*.[[ArrayBufferByteLength]].

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

    Note

    This step might throw an exception if there is not enough memory available to create such an ArrayBuffer object.

15. Otherwise, if *serialized*.[[Type]] is "ResizableArrayBuffer", then set *value* to a new ArrayBuffer object in *targetRealm* whose [[ArrayBufferData]] internal slot value is *serialized*.[[ArrayBufferData]], whose [[ArrayBufferByteLength]] internal slot value is *serialized*.[[ArrayBufferByteLength]], and whose [[ArrayBufferMaxByteLength]] internal slot value is *serialized*.[[ArrayBufferMaxByteLength]].

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

    Note

    This step might throw an exception if there is not enough memory available to create such an ArrayBuffer object.

16. Otherwise, if *serialized*.[[Type]] is "ArrayBufferView":

    1. Let *deserializedArrayBuffer* be ? StructuredDeserialize[p128](*serialized*.[[ArrayBufferSerialized]], *targetRealm*, *memory*).

    2. If *serialized*.[[Constructor]] is "DataView", then set *value* to a new DataView object in *targetRealm* whose [[ViewedArrayBuffer]] internal slot value is *deserializedArrayBuffer*, whose [[ByteLength]] internal slot value is *serialized*.[[ByteLength]], and whose [[ByteOffset]] internal slot value is *serialized*.[[ByteOffset]].

    3. Otherwise, set *value* to a new typed array object in *targetRealm*, using the constructor given by *serialized*.[[Constructor]], whose [[ViewedArrayBuffer]] internal slot value is *deserializedArrayBuffer*, whose [[TypedArrayName]] internal slot value is *serialized*.[[Constructor]], whose [[ByteLength]] internal slot value is *serialized*.[[ByteLength]], whose [[ByteOffset]] internal slot value is *serialized*.[[ByteOffset]], and whose [[ArrayLength]] internal slot value is *serialized*.[[ArrayLength]].

17. Otherwise, if *serialized*.[[Type]] is "Map":

    1. Set *value* to a new Map object in *targetRealm* whose [[MapData]] internal slot value is a new empty List.

    2. Set *deep* to true.

18. Otherwise, if *serialized*.[[Type]] is "Set":

    1. Set *value* to a new Set object in *targetRealm* whose [[SetData]] internal slot value is a new empty List.

    2. Set *deep* to true.

19. Otherwise, if *serialized*.[[Type]] is "Array":

    1. Let *outputProto* be *targetRealm*.[[Intrinsics]].[[%Array.prototype%]].

    2. Set *value* to ! ArrayCreate(*serialized*.[[Length]], *outputProto*).

    3. Set *deep* to true.

20. Otherwise, if *serialized*.[[Type]] is "Object":

    1. Set *value* to a new Object in *targetRealm*.

    2. Set *deep* to true.

21. Otherwise, if *serialized*.[[Type]] is "Error":

    1. Let *prototype* be %Error.prototype%.

    2. If *serialized*.[[Name]] is "EvalError", then set *prototype* to %EvalError.prototype%[p58].

3. If *serialized*.[[Name]] is "RangeError", then set *prototype* to %RangeError.prototype%[p58].
4. If *serialized*.[[Name]] is "ReferenceError", then set *prototype* to %ReferenceError.prototype%[p58].
5. If *serialized*.[[Name]] is "SyntaxError", then set *prototype* to %SyntaxError.prototype%[p58].
6. If *serialized*.[[Name]] is "TypeError", then set *prototype* to %TypeError.prototype%[p58].
7. If *serialized*.[[Name]] is "URIError", then set *prototype* to %URIError.prototype%[p58].
8. Let *message* be *serialized*.[[Message]].
9. Set *value* to OrdinaryObjectCreate(*prototype*, « [[ErrorData]], [[Stack]] »).
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. Set *value*.[[Stack]] to *serialized*.[[Stack]].
13. Any interesting accompanying data attached to *serialized* should be deserialized and attached to *value*.

22. 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.
23. Set *memory*[*serialized*] to *value*.
24. If *deep* is true:
    1. If *serialized*.[[Type]] is "Map":
       1. For each Record { [[Key]], [[Value]] } *entry* of *serialized*.[[MapData]]:
          1. Let *deserializedKey* be ? StructuredDeserialize[p128](*entry*.[[Key]], *targetRealm*, *memory*).
          2. Let *deserializedValue* be ? StructuredDeserialize[p128](*entry*.[[Value]], *targetRealm*, *memory*).
          3. Append { [[Key]]: *deserializedKey*, [[Value]]: *deserializedValue* } to *value*.[[MapData]].
    2. Otherwise, if *serialized*.[[Type]] is "Set":
       1. For each *entry* of *serialized*.[[SetData]]:
          1. Let *deserializedEntry* be ? StructuredDeserialize[p128](*entry*, *targetRealm*, *memory*).
          2. Append *deserializedEntry* to *value*.[[SetData]].
    3. Otherwise, if *serialized*.[[Type]] is "Array" or "Object":
       1. For each Record { [[Key]], [[Value]] } *entry* of *serialized*.[[Properties]]:
          1. Let *deserializedValue* be ? StructuredDeserialize[p128](*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[p122] for the interface identified by *serialized*.[[Type]], given *serialized*, *value*, and *targetRealm*.

          The deserialization steps[p122] may need to perform a **sub-deserialization**. This is an operation which takes as input a previously-serialized Record *subSerialized*, and returns

StructuredDeserialize[p128](*subSerialized*, *targetRealm*, *memory*). (In other words, a sub-deserialization[p130] is a specialization of StructuredDeserialize[p128] to be consistent within this invocation.)

25. Return *value*.

2.7.7 StructuredSerializeWithTransfer (*value*, *transferList*) §[p131]

1. Let *memory* be an empty map.

   Note

   In addition to how it is used normally by StructuredSerializeInternal[p124], in this algorithm memory *is also used to ensure that StructuredSerializeInternal[p124] 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]][p124] 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[p124] needs to be able to throw first.*

3. Let *serialized* be ? StructuredSerializeInternal[p124](*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]][p124] internal slot and *transferable*.[[Detached]][p124] is true, then throw a "DataCloneError" DOMException.
   3. Let *dataHolder* be *memory*[*transferable*].
   4. If *transferable* has an [[ArrayBufferData]] internal slot:

      1. If *transferable* has an [[ArrayBufferMaxByteLength]] internal slot:

         1. Set *dataHolder*.[[Type]] to "ResizableArrayBuffer".
         2. Set *dataHolder*.[[ArrayBufferData]] to *transferable*.[[ArrayBufferData]].
         3. Set *dataHolder*.[[ArrayBufferByteLength]] to *transferable*.[[ArrayBufferByteLength]].
         4. Set *dataHolder*.[[ArrayBufferMaxByteLength]] to *transferable*.[[ArrayBufferMaxByteLength]].

      2. Otherwise:

         1. Set *dataHolder*.[[Type]] to "ArrayBuffer".
         2. Set *dataHolder*.[[ArrayBufferData]] to *transferable*.[[ArrayBufferData]].
         3. Set *dataHolder*.[[ArrayBufferByteLength]] to *transferable*.[[ArrayBufferByteLength]].

      3. Perform ? DetachArrayBuffer(*transferable*).

Note

Specifications can use the [[ArrayBufferDetachKey]] internal slot to prevent ArrayBuffers from being detached. This is used in WebAssembly JavaScript Interface*, for example. [WASMJS]*[p1587]

5. Otherwise:
    1. Assert: *transferable* is a platform object that is a transferable object[p123].
    2. Let *interfaceName* be the identifier of the primary interface of *transferable*.
    3. Set *dataHolder*.[[Type]] to *interfaceName*.
    4. Perform the appropriate transfer steps[p123] for the interface identified by *interfaceName*, given *transferable* and *dataHolder*.
    5. Set *transferable*.[[Detached]][p124] to true.
6. Append *dataHolder* to *transferDataHolders*.

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

2.7.8 StructuredDeserializeWithTransfer (*serializeWithTransferResult*, *targetRealm*) §[p132]

1. Let *memory* be an empty map.

Note

Analogous to StructuredSerializeWithTransfer[p131], in addition to how it is used normally by StructuredDeserialize[p128], in this algorithm memory *is also used to ensure that StructuredDeserialize[p128] 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, if *transferDataHolder*.[[Type]] is "ResizableArrayBuffer", then set *value* to a new ArrayBuffer object in *targetRealm* whose [[ArrayBufferData]] internal slot value is *transferDataHolder*.[[ArrayBufferData]], whose [[ArrayBufferByteLength]] internal slot value is *transferDataHolder*.[[ArrayBufferByteLength]], and whose [[ArrayBufferMaxByteLength]] internal slot value is *transferDataHolder*.[[ArrayBufferMaxByteLength]].

    Note

    For the same reason as the previous step, this step is also unlikely to throw an exception.

    4. 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[p123] for the interface identified by *interfaceName* given

transferDataHolder and *value*.

5. Set *memory*[*transferDataHolder*] to *value*.

6. Append *value* to *transferredValues*.

4. Let *deserialized* be ? StructuredDeserialize[p128](*serializeWithTransferResult*.[[Serialized]], *targetRealm*, *memory*).

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

2.7.9 Performing serialization and transferring from other specifications §[p133]

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[p131]

StructuredDeserializeWithTransfer[p132]

Cloning a value to another 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

`messagePort.postMessage()`[p1302] uses this pair of abstract operations, as the destination realm is not known until the `MessagePort`[p1299] has been shipped[p1300].

StructuredSerialize[p128]

StructuredSerializeForStorage[p128]

StructuredDeserialize[p128]

Creating a 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[p128] 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[p122] when the *forStorage* argument is true.

Example

`history.pushState()`[p989] and `history.replaceState()`[p989] use StructuredSerializeForStorage[p128] on author-supplied state objects, storing them as serialized state[p1053] in the appropriate session history entry[p1053]. Then, StructuredDeserialize[p128] is used so that the `history.state`[p989] property can return a clone of the originally-supplied state object.

Example

`broadcastChannel.postMessage()`[p1304] uses StructuredSerialize[p128] on its input, then uses StructuredDeserialize[p128] 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[p128] on its input and StructuredDeserialize[p128] 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.

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[p1166] and prepare to run a callback[p1148] before invoking StructuredSerialize[p128], StructuredSerializeForStorage[p128], or StructuredSerializeWithTransfer[p131] 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[p1146] and incumbent[p1146] concepts being properly set up.

Example

`window.postMessage()`[p1296] performs StructuredSerializeWithTransfer[p131] 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[p1166] and prepare to run a callback[p1148].

Example

In contrast, a hypothetical API that used StructuredSerialize[p128] to serialize some author-supplied object periodically, directly from a task[p1194] on the event loop[p1193], 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.

2.7.10 Structured cloning API §[p134]

For web developers (non-normative)

***result* = self.structuredClone[p1134](*value*[, { transfer[p1299] }])**

Takes the input value and returns a deep copy by performing the structured clone algorithm. Transferable objects[p123] listed in the `transfer`[p1299] array are transferred, not just cloned, meaning that they are no longer usable in the input value.

Throws a "`DataCloneError`" `DOMException` if any part of the input value is not serializable[p122].

The **`structuredClone(value, options)`** method steps are:

1. Let *serialized* be ? StructuredSerializeWithTransfer[p131](*value*, *options*["`transfer`[p1299]"]).
2. Let *deserializeRecord* be ? StructuredDeserializeWithTransfer[p132](*serialized*, this's relevant realm[p1151]).
3. Return *deserializeRecord*.[[Deserialized]].

3 Semantics, structure, and APIs of HTML documents §p135

3.1 Documents §p135

Every XML and HTML document in an HTML UA is represented by a Document[p135] object. [DOM][p1582]

The Document[p135] object's **URL** is defined in *DOM*. It is initially set when the Document[p135] object is created, but can change during the lifetime of the Document[p135] object; for example, it changes when the user navigates[p1063] to a fragment[p1071] on the page and when the pushState()[p989] method is called with a new URL. [DOM][p1582]

⚠Warning!

Interactive user agents typically expose the Document[p135] 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[p135] object's **origin** is defined in *DOM*. It is initially set when the Document[p135] object is created, and can change during the lifetime of the Document[p135] only upon setting document.domain[p942]. A Document[p135]'s origin can differ from the origin of its URL; for example when a child navigable[p1039] is created[p1039], its active document[p1036]'s origin is inherited from its parent[p1035]'s active document[p1036]'s origin, even though its active document[p1036]'s URL is about:blank[p54]. [DOM][p1582]

When a Document[p135] is created by a script[p1152] using the createDocument() or createHTMLDocument() methods, the Document[p135] is ready for post-load tasks[p1459] immediately.

The document's referrer is a string (representing a URL) that can be set when the Document[p135] is created. If it is not explicitly set, then its value is the empty string.

3.1.1 The Document[p135] object §p135

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

IDL

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

[LegacyOverrideBuiltIns]
partial interface Document {
  static Document parseHTMLUnsafe((TrustedHTML or DOMString) html, optional ParseHTMLUnsafeOptions
options = {});
  static Document parseHTML(DOMString html, optional SetHTMLOptions options = {});

  // 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 HTMLElement? body;
  readonly attribute HTMLHeadElement? head;
  [SameObject] readonly attribute HTMLCollection images;
  [SameObject] readonly attribute HTMLCollection embeds;
  [SameObject] readonly attribute HTMLCollection plugins;
```

```
  [SameObject] readonly attribute HTMLCollection links;
  [SameObject] readonly attribute HTMLCollection forms;
  [SameObject] readonly attribute HTMLCollection scripts;
  NodeList getElementsByName(DOMString elementName);
  readonly attribute 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((TrustedHTML or DOMString)... text);
  [CEReactions] undefined writeln((TrustedHTML or 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 queryCommandIndeterm(DOMString commandId);
  boolean queryCommandState(DOMString commandId);
  boolean queryCommandSupported(DOMString commandId);
  DOMString queryCommandValue(DOMString commandId);
  readonly attribute boolean hidden;
  readonly attribute DocumentVisibilityState visibilityState;

  // special event handler IDL attributes that only apply to Document objects
  [LegacyLenientThis] attribute EventHandler onreadystatechange;
  attribute EventHandler onvisibilitychange;

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

Each Document[p135] has a **policy container** (a policy container[p960]), initially a new policy container, which contains policies which apply to the Document[p135].

Each Document[p135] has a **permissions policy**, which is a permissions policy, which is initially empty.

Each Document[p135] has a **module map**, which is a module map[p1189], initially empty.

Each Document[p135] has an **opener policy**, which is an opener policy[p945], initially a new opener policy.

Each Document[p135] has an **is initial `about:blank`**, which is a boolean, initially false.

Each Document[p135] has a **during-loading navigation ID for WebDriver BiDi**, which is a navigation ID[p1062] or null, initially null.

Note

As the name indicates, this is used for interfacing with the WebDriver BiDi *specification, which needs to be informed about certain occurrences during the early parts of the* Document[p135]*'s lifecycle, in a way that ties them to the original navigation ID[p1062] used when the navigation that created this* Document[p135] *was the ongoing navigation[p1076]. This eventually gets set back to null, after* WebDriver BiDi *considers the loading process to be finished. [BIDI][p1579]*

Each Document[p135] has an **about base URL**, which is a URL or null, initially null.

Note

This is only populated for `"about:"`*-schemed* Document[p135]*s.*

Each Document[p135] has a **bfcache blocking details**, which is a set of not restored reason details[p1031], initially empty.

Each Document[p135] has an **open dialogs list**, which is a list of dialog[p675] elements, initially empty.

3.1.2 The DocumentOrShadowRoot[p137] interface §[p137]

DOM defines the DocumentOrShadowRoot mixin, which this specification extends.

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

3.1.3 Ancestor origins §[p137]

A Document[p135] object has an associated **internal ancestor origin objects list**, initially null.

The **internal ancestor origin objects list creation steps** given a Document[p135] object *document* and a referrer policy *referrerPolicy* are:

1. Let *output* be « ».
2. Let *parentDoc* be *document*'s container document[p1039].
3. If *parentDoc* is null, then return *output*.
4. Assert: *parentDoc* is fully active[p1051].
5. Let *ancestorOrigins* be *parentDoc*'s internal ancestor origin objects list[p137].
6. Assert: *ancestorOrigins* is not null.
7. Let *masked* be false.
8. If *referrerPolicy* is "`no-referrer`", then set *masked* to true.
9. Otherwise, if *referrerPolicy* is "`same-origin`" and *parentDoc*'s origin is not same origin[p940] with *document*'s origin, then set *masked* to true.

   Note

   Since mixed content checks prevent non-secure context[p1152] environments[p1143] in secure context[p1152] environments[p1143], there's no need to check for secure contexts[p1152] for "`strict-origin`", "`strict-origin-when-cross-origin`", *and* "`no-referrer-when-downgrade`". *The* "`origin`" *and* "`origin-when-cross-origin`" *values also don't need special treatment since we're at most exposing an origin here.*
10. If *masked* is true, then append a new opaque origin[p939] to *output*.
11. Otherwise, append *parentDoc*'s origin to *output*.
12. For each *ancestorOrigin* of *ancestorOrigins*:
    1. If *masked* is true and *ancestorOrigin* is same origin[p940] with *parentDoc*'s origin, then append a new opaque origin[p939] to *output* and continue.
    2. Append *ancestorOrigin* to *output* and set *masked* to false.

       Note

       Setting masked *to false here doesn't imply that all further ancestors' origins are necessarily exposed. They might have been previously masked when an ancestor document ran these steps, and the resulting list is used as a starting point when a child document is created (see step 5 above).*
13. Return *output*.

A Document[p135] object has an associated **ancestor origins list**, initially null.

The **ancestor origins list creation steps** given a Document[p135] object *document* are:

1. Let *ancestorOrigins* be *document*'s internal ancestor origin objects list[p137].
2. Assert: *ancestorOrigins* is not null.
3. Let *output* be « ».
4. For each *origin* of *ancestorOrigins*:
    1. Append the serialization[p939] of *origin* to *output*.
5. Return a new DOMStringList[p121] object whose associated list is *output*.

Example

Consider a top-level browsing context[p1049] document with the URL `https://a.example/top`:

```
<!doctype html>
<title>top</title>
<iframe referrerpolicy="no-referrer" src="https://a.example/child"></iframe>
```

The child document:

```
<!doctype html>
<title>child</title>
<iframe src="https://b.example/grandchild"></iframe>
<script>
 console.log([...location.ancestorOrigins]);
</script>
```

The grandchild document:

```
<!doctype html>
<title>grandchild</title>
<script>
 console.log([...location.ancestorOrigins]);
</script>
```

When the child's Document[p135] object is created, its parent's origin is masked because the iframe[p404] element's referrerpolicy[p412] attribute's value is "`no-referrer`", even though the documents are same origin[p940]. The value logged is « "`null`" ».

When the grandchild's Document[p135] object is created, the child's Document[p135] object's internal ancestor origin objects list[p137] (which contains a single opaque origin[p939]) is used as a starting point, and the child's origin is added to the list. The value logged is thus « "`https://a.example`", "`null`" ».

Example

If we consider the previous example but the iframe[p404] element in the top document has no referrerpolicy[p412] attribute, and instead the iframe[p404] element in the child document has `referrerpolicy="no-referrer"`.

In this case, the top document's origin is added to the child's Document[p135] object's internal ancestor origin objects list[p137]. The value logged is « "`https://a.example`" ».

For the grandchild, the child's origin is masked as an opaque origin[p939]. Since the top document's origin is same origin[p940] with the child document's origin, it is also masked. The value logged is thus « "`null`", "`null`" ».

Example

In this example, we show that the top-level document's origin is not masked even though there's a descendant same-origin document whose origin *is* masked, since there's a cross-origin document in between.

- Top-level with URL `https://a.example/top`

  ```
  <iframe src="https://b.example/child"></iframe>
  ```

  The ancestor origins list[p138] associated list is « ».

- Child with URL `https://b.example/child`

  ```
  <iframe src="https://a.example/grandchild"></iframe>
  ```

  The ancestor origins list[p138] associated list is « "https://a.example" ».

- Grandchild with URL `https://a.example/grandchild`

  ```
  <iframe src="https://c.example/great-grandchild"
          referrerpolicy="no-referrer"></iframe>
  ```

  The ancestor origins list[p138] associated list is « "https://b.example", "https://a.example" ».

- Great-grandchild with URL `https://c.example/great-grandchild`

  The ancestor origins list[p138] associated list is « "null", "https://b.example", "https://a.example" ». The top-level origin is not masked.

Example

If we consider the previous example but the top-level document's iframe[p404] also has `referrerpolicy="no-referrer"`, then the resulting ancestor origins list[p138] associated list for each document would be:

- Top-level: « »
- Child: « "null" »
- Grandchild: « "https://b.example", "null" »
- Great-grandchild: « "null", "https://b.example", "null" ». The top-level origin is masked because it was masked when the child document ran the internal ancestor origin objects list creation steps[p137], and that result is passed on to the grandchild document's internal ancestor origin objects list creation steps[p137], and so on.

3.1.4 Resource metadata management §[p139]

For web developers (non-normative)

***document*.referrer**[p139]

Returns the URL of the Document[p135] from which the user navigated to this one, unless it was blocked or there was no such document, in which case it returns the empty string.

The noreferrer[p338] link type can be used to block the referrer.

The **referrer** attribute must return the document's referrer[p135].

For web developers (non-normative)

***document*.cookie**[p140] **[= *value*]**

Returns the HTTP cookies that apply to the Document[p135]. If there are no cookies or cookies can't be applied to this resource, the empty string will be returned.

Can be set, to add a new cookie to the element's set of HTTP cookies.

If the contents are sandboxed into an opaque origin[p957] (e.g., in an iframe[p404] with the sandbox[p409] attribute), a

"SecurityError" DOMException will be thrown on getting and setting.

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

Note

Using the synchronous document.cookie[p140] API can be a source of performance issues. The Cookie Store API *can be used instead, as it provides an asynchronous way to handle cookies to avoid performance issues. See the Cookie Store API introduction for more information. [COOKIESTORE][p1580]*

A Document[p135] object that falls into one of the following conditions is a **cookie-averse Document object**:

- A Document[p135] object whose browsing context[p1046] is null.
- A Document[p135] whose URL's scheme is not an HTTP(S) scheme.

On getting, if the document is a cookie-averse Document object[p140], then the user agent must return the empty string. Otherwise, if the Document[p135]'s origin is an opaque origin[p939], 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][p1580]

On setting, if the document is a cookie-averse Document object[p140], then the user agent must do nothing. Otherwise, if the Document[p135]'s origin is an opaque origin[p939], 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][p1580] [ENCODING][p1582]

Note

Since the cookie[p140] 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[p140] 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)

***document*.lastModified[p140]**

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[p135]'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.

3.1.5 Reporting document loading status

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**, a string, initially "complete".

Note

For Document objects created via the create and initialize a Document object algorithm, this will be immediately reset to "loading" before any script can observe the value of document.readyState. This default applies to other cases such as initial about:blank Documents or Documents without a browsing context.

The **readyState** getter steps are to return this's current document readiness.

To **update the current document readiness** for Document *document* to *readinessValue*:

1. If *document*'s current document readiness equals *readinessValue*, then return.
2. Set *document*'s current document readiness to *readinessValue*.
3. If *document* is associated with an HTML parser:
    1. Let *now* be the current high resolution time given *document*'s relevant global object.
    2. If *readinessValue* is "complete", and *document*'s load timing info's DOM complete time is 0, then set *document*'s load timing info's DOM complete time to *now*.
    3. Otherwise, if *readinessValue* is "interactive", and *document*'s load timing info's DOM interactive time is 0, then set *document*'s load timing info's DOM interactive time to *now*.
4. Fire an event named readystatechange at *document*.

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.

A Document has a document load timing info **load timing info**.

A Document has a document unload timing info **previous document unload timing**.

A Document[p135] has a boolean **was created via cross-origin redirects**, initially false.

The **document load timing info** struct has the following items:

navigation start time **(default 0)**
: A number

DOM interactive time **(default 0)**
DOM content loaded event start time **(default 0)**
DOM content loaded event end time **(default 0)**
DOM complete time **(default 0)**
load event start time **(default 0)**
load event end time **(default 0)**
: DOMHighResTimeStamp values

The **document unload timing info** struct has the following items:

unload event start time **(default 0)**
unload event end time **(default 0)**
: DOMHighResTimeStamp values

3.1.6 Render-blocking mechanism §[p142]

Each Document[p135] has a **render-blocking element set**, a set of elements, initially the empty set.

A Document[p135] *document* **allows adding render-blocking elements** if *document*'s content type is "text/html[p1546]" and the body element[p144] of *document* is null.

A Document[p135] *document* is **render-blocked** if both of the following are true:

- *document*'s render-blocking element set[p142] is non-empty, or *document* allows adding render-blocking elements[p142].
- The current high resolution time given *document*'s relevant global object[p1151] has not exceeded an implementation-defined timeout value.

An element *el* is **render-blocking** if *el*'s node document *document* is render-blocked[p142], and *el* is in *document*'s render-blocking element set[p142].

To **block rendering** on an element *el*:

1. Let *document* be *el*'s node document.
2. If *document* allows adding render-blocking elements[p142], then append *el* to *document*'s render-blocking element set[p142].

To **unblock rendering** on an element *el*:

1. Let *document* be *el*'s node document.
2. Remove *el* from *document*'s render-blocking element set[p142].

Whenever a render-blocking[p142] element *el* becomes browsing-context disconnected[p47], or *el*'s blocking attribute[p107]'s value is changed so that *el* is no longer potentially render-blocking[p107], then unblock rendering[p142] on *el*.

3.1.7 DOM tree accessors §[p142]

The `html` element of a document is its document element, if it's an html[p179] element, and null otherwise.

For web developers (non-normative)

***document*.head**[p143]

Returns the head element[p143].

The head element of a document is the first head[p180] element that is a child of the html element[p142], if there is one, or null otherwise.

The **head** attribute, on getting, must return the head element[p143] of the document (a head[p180] element or null).

For web developers (non-normative)

***document*.title**[p143] **[= *value*]**

Returns the document's title, as given by the title element[p143] for HTML and as given by the SVG title element for SVG.

Can be set, to update the document's title. If there is no appropriate element to update, the new value is ignored.

The title element of a document is the first title[p181] element in the document (in tree order), if there is one, or null otherwise.

The **title** attribute must, on getting, run the following algorithm:

✔ MDN

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[p143], or the empty string if the title element[p143] 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[p143] is null and the head element[p143] is null, then return.
2. If the title element[p143] is non-null, let *element* be the title element[p143].
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[p143].
4. String replace all with the given value within *element*.

↪ **Otherwise**

Do nothing.

For web developers (non-normative)

***document*.body**[p144] **[= *value*]**

Returns the body element[p144].

Can be set, to replace the body element[p144].

If the new value is not a body[p212] or frameset[p1537] element, this will throw a "HierarchyRequestError" DOMException.

The body element of a document is the first of the html element[p142]'s children that is either a body[p212] element or a frameset[p1537] element, or null if there is no such element.

The **body** attribute, on getting, must return the body element[p144] of the document (either a body[p212] element, a frameset[p1537] element, or null). On setting, the following algorithm must be run:

1. If the new value is not a body[p212] or frameset[p1537] element, then throw a "HierarchyRequestError" DOMException.
2. Otherwise, if the new value is the same as the body element[p144], return.
3. Otherwise, if the body element[p144] is not null, then replace the body element[p144] with the new value within the body element[p144]'s parent and return.
4. Otherwise, if there is no document element, throw a "HierarchyRequestError" DOMException.
5. Otherwise, the body element[p144] is null, but there's a document element. Append the new value to the document element.

Note

The value returned by the body[p144] getter is not always the one passed to the setter.

Example

In this example, the setter successfully inserts a body[p212] element (though this is non-conforming since SVG does not allow a body[p212] as child of SVG svg). However the getter will return null because the document element is not html[p179].

```
<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>
```

For web developers (non-normative)

***document*.images**[p144]

Returns an HTMLCollection of the img[p359] elements in the Document[p135].

***document*.embeds**[p144]

***document*.plugins**[p144]

Returns an HTMLCollection of the embed[p413] elements in the Document[p135].

***document*.links**[p144]

Returns an HTMLCollection of the a[p267] and area[p489] elements in the Document[p135] that have href[p314] attributes.

***document*.forms**[p144]

Returns an HTMLCollection of the form[p532] elements in the Document[p135].

***document*.scripts**[p145]

Returns an HTMLCollection of the script[p685] elements in the Document[p135].

The **images** attribute must return an HTMLCollection rooted at the Document[p135] node, whose filter matches only img[p359] elements.

The **embeds** attribute must return an HTMLCollection rooted at the Document[p135] node, whose filter matches only embed[p413] elements.

The **plugins** attribute must return the same object as that returned by the embeds[p144] attribute.

The **links** attribute must return an HTMLCollection rooted at the Document[p135] node, whose filter matches only a[p267] elements with href[p314] attributes and area[p489] elements with href[p314] attributes.

The **forms** attribute must return an HTMLCollection rooted at the Document[p135] node, whose filter matches only form[p532] elements.

The **scripts** attribute must return an HTMLCollection rooted at the Document[p135] node, whose filter matches only script[p685] elements.

For web developers (non-normative)

***collection* = *document*.getElementsByName[p145](*name*)**

Returns a NodeList of elements in the Document[p135] that have a name attribute with the value *name*.

The **getElementsByName(*elementName*)** method steps are to return a live[p48] NodeList containing all the HTML elements[p46] in that document that have a name attribute whose value is identical to the *elementName* argument, in tree order. When the method is invoked on a Document[p135] 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[p145]**

Returns the script[p685] element, or the SVG script element, that is currently executing, as long as the element represents a classic script[p1153]. 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[p135] is not currently executing a script[p685] or SVG script element (e.g., because the running script is an event handler, or a timeout), or if the currently executing script[p685] or SVG script element represents a module script[p1153].

The **currentScript** attribute, on getting, must return the value to which it was most recently set. When the Document[p135] is created, the currentScript[p145] must be initialized to null.

Note

This API has fallen out of favor in the implementer and standards community, as it globally exposes script[p685] or SVG script elements. As such, it is not available in newer contexts, such as when running module scripts[p1153] 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[p135] interface supports named properties. The supported property names of a Document[p135] object *document* at any moment consist of the following, for each element *element* in a document tree with *document* as its root, in tree order, ignoring later duplicates:

- the value of *element*'s id[p162] content attribute, if *element* is a named element[p145] with that value as its name; and then
- the value of *element*'s name content attribute, if *element* is a named element[p145] with that value as its name.

To determine the value of a named property *name* for a Document[p135], the user agent must return the value obtained using the following steps:

1. Let *elements* be the list of named elements[p145] with the name *name* that are in a document tree with the Document[p135] as their root.

   Note

   There will be at least one such element, since the algorithm would otherwise not have been invoked by Web IDL.

2. If *elements* has only one element, and that element is an iframe[p404] element, and that iframe[p404] element's content navigable[p1038] is not null, then return the active WindowProxy[p1036] of the element's content navigable[p1038].
3. Otherwise, if *elements* has only one element, return that element.
4. Otherwise, return an HTMLCollection rooted at the Document[p135] node, whose filter matches only named elements[p145] with the name *name*.

An element is a **named element** with the name *name* if *name* is not the empty string and any of the following are true:

- it is an embed[p413], form[p532], iframe[p404], img[p359], or object[p416] element whose name content attribute's value is *name*;

- it is an object[p416] element whose id[p162] content attribute's value is *name*; or
- it is an img[p359] element whose id[p162] content attribute's value is *name*, and that has a non-empty name content attribute present also.

Note

The dir[p170] attribute on the Document[p135] interface is defined along with the dir[p167] content attribute.

3.2 Elements §[p14]

3.2.1 Semantics §[p14]

Elements, attributes, and attribute values in HTML are defined (by this specification) to have certain meanings (semantics). For example, the ol[p247] element represents an ordered list, and the lang[p165] 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:

```
<!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 section).

```
<body>
 <h1>ACME Corporation</h1>
 <h2>The leaders in arbitrary fast delivery since 1920</h2>
 ...
```

The hgroup[p225] element can be used for these kinds of situations:

```
<body>
 <hgroup>
  <h1>ACME Corporation</h1>
  <p>The leaders in arbitrary fast delivery since 1920</p>
 </hgroup>
 ...
```

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[p275] 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/"><cite>Ernest</cite></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>
```

Authors must not use elements, attributes, or attribute values that are not permitted by this specification or other applicable

specifications[p77], as doing so makes it significantly harder for the language to be extended in the future.

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:

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

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

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

DOM nodes whose node document's browsing context[p1046] is null are exempt from all document conformance requirements other than the HTML syntax[p1356] requirements and XML syntax[p1484] requirements.

Example

In particular, the `template`[p705] element's template contents[p707]'s node document's browsing context[p1046] is null. For example, the content model[p154] requirements and attribute value microsyntax requirements do not apply to a `template`[p705] element's template contents[p707]. In this example an `img`[p359] element has attribute values that are placeholders that would be invalid outside a `template`[p705] element.

```
<template>
 <article>
  <img src="{{src}}" alt="{{alt}}">
  <h1></h1>
 </article>
</template>
```

However, if the above markup were to omit the `</h1>` end tag, that would be a violation of the HTML syntax[p1356], and would thus be flagged as an error by conformance checkers.

Through scripting and using other mechanisms, the values of attributes, text, and indeed the entire structure of the document may change dynamically while a user agent is processing it. The semantics of a document at an instant in time are those represented by the state of the document at that instant in time, and the semantics of a document can therefore change over time. User agents must update their presentation of the document as this occurs.

Example

HTML has a `progress`[p614] element that describes a progress bar. If its "value" attribute is dynamically updated by a script, the UA would update the rendering to show the progress changing.

3.2.2 Elements in the DOM §[p14]

The nodes representing HTML elements[p46] 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[p46] 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`[p247] 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`[p162] attribute to the element, and then creating a hyperlink[p313] with that `id`[p162] attribute's value as the fragment[p1071] for the hyperlink[p313]'s `href`[p314] attribute value. Hyperlinks are not necessary for a reference, however; any manner of referring to the element in question will suffice.

Example

Consider the following figure[p259] element, which is given an id[p162] attribute:

```
<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[p313]-based reference[p148] could be created using the a[p267] element, like so:

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

However, there are many other ways of referencing[p148] the figure[p259] 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[p259])

The basic interface, from which all the HTML elements[p46]' interfaces inherit, and which must be used by elements that have no additional requirements, is the HTMLElement[p149] interface.

✓ MDN

IDL

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

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

  // user interaction
  [CEReactions] attribute (boolean or unrestricted double or DOMString)? hidden;
  [CEReactions, Reflect] attribute boolean inert;
  undefined click();
  [CEReactions, Reflect] attribute DOMString accessKey;
  readonly attribute DOMString accessKeyLabel;
  [CEReactions] attribute boolean draggable;
  [CEReactions] attribute boolean spellcheck;
  [CEReactions, ReflectSetter] attribute DOMString writingSuggestions;
  [CEReactions, ReflectSetter] attribute DOMString autocapitalize;
  [CEReactions] attribute boolean autocorrect;

  [CEReactions] attribute [LegacyNullToEmptyString] DOMString innerText;
  [CEReactions] attribute [LegacyNullToEmptyString] DOMString outerText;

  ElementInternals attachInternals();

  // The popover API
  undefined showPopover(optional ShowPopoverOptions options = {});
  undefined hidePopover();
  boolean togglePopover(optional (TogglePopoverOptions or boolean) options = {});
  [CEReactions] attribute DOMString? popover;

  [CEReactions, Reflect, ReflectRange=(0, 8)] attribute unsigned long headingOffset;
```

```
  [CEReactions, Reflect] attribute boolean headingReset;
};

dictionary ShowPopoverOptions {
  HTMLElement source;
};

dictionary TogglePopoverOptions : ShowPopoverOptions {
  boolean force;
};

HTMLElement includes GlobalEventHandlers;
HTMLElement includes ElementContentEditable;
HTMLElement includes HTMLOrSVGOrMathMLElement;

[Exposed=Window]
interface HTMLUnknownElement : HTMLElement {
  // Note: intentionally no [HTMLConstructor]
};
```

The HTMLElement[p149] 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[p1530], bgsound[p1530], blink[p1531], isindex[p1530], keygen[p1530], multicol[p1531], nextid[p1530], or spacer[p1531], then return HTMLUnknownElement[p150].
2. If *name* is acronym[p1530], basefont[p1531], big[p1531], center[p1531], nobr[p1531], noembed[p1530], noframes[p1530], plaintext[p1530], rb[p1531], rtc[p1531], strike[p1531], or tt[p1531], then return HTMLElement[p149].
3. If *name* is listing[p1530] or xmp[p1531], then return HTMLPreElement[p243].
4. Otherwise, if this specification defines an interface appropriate for the element type[p46] corresponding to the local name *name*, then return that interface.
5. If other applicable specifications[p77] define an appropriate interface for *name*, then return the interface they define.
6. If *name* is a valid custom element name[p797], then return HTMLElement[p149].
7. Return HTMLUnknownElement[p150].

Note

The use of HTMLElement[p149] instead of HTMLUnknownElement[p150] in the case of valid custom element names[p797] is done to ensure that any potential future upgrades[p803] only cause a linear transition of the element's prototype chain, from HTMLElement[p149] to a subclass, instead of a lateral one, from HTMLUnknownElement[p150] to an unrelated subclass.

Features shared between HTML, SVG and MathML elements use the HTMLOrSVGOrMathMLElement[p150] interface mixin: [SVG][p1586] [MATHML][p1584]

IDL

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

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

Example

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

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

3.2.3 HTML element constructors §p151 1

To support the custom elements[p785] 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][p151] 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][p151] extended attribute have the following overridden constructor steps:

1. 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:

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

   In this case, during the execution of HTMLButtonElement[p585] (either explicitly, as in (1), or implicitly, as in (2)), both the active function object and NewTarget are HTMLButtonElement[p585]. If this check was not present, it would be possible to create an instance of HTMLButtonElement[p585] whose local name was `bad-1`.

2. Let *registry* be null.

3. If the surrounding agent's active custom element constructor map[p798][NewTarget] exists, then set *registry* to the surrounding agent's active custom element constructor map[p798][NewTarget].

4. Otherwise, set *registry* to the current global object[p1151]'s associated Document[p966]'s custom element registry.

5. Let *definition* be the item in *registry*'s custom element definition set[p799] with constructor[p798] equal to NewTarget. If there is no such item, then throw a TypeError.

   Note

   Since there can be no item in registry's *custom element definition set[p799] with a constructor[p798] of undefined, this step also prevents HTML element constructors from being called as functions (since in that case NewTarget will be undefined).*

6. Let *isValue* be null.

7. If *definition*'s local name[p798] is equal to *definition*'s name[p798] (i.e., *definition* is for an autonomous custom element[p796]):

   1. If the active function object is not HTMLElement[p149], 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[p149] class:

      ```
      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[p238], not HTMLElement[p149].

8. Otherwise (i.e., if *definition* is for a customized built-in element[p796]):

    1. Let *valid local names* be the list of local names for elements defined in this specification or in other applicable specifications[p77] that use the active function object as their element interface.

    2. If *valid local names* does not contain *definition*'s local name[p798], 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:

    ```
    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 `q`[p277] and `blockquote`[p244], but *definition*'s local name[p798] is `p`[p238], which is not in that list.

    3. Set *isValue* to *definition*'s name[p798].

9. If *definition*'s construction stack[p798] is empty:

    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 and NewTarget.

    2. Set *element*'s node document to the current global object[p1151]'s associated `Document`[p966].

    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[p798].

    6. Set *element*'s custom element registry to *registry*.

    7. Set *element*'s custom element state to "`custom`".

    8. Set *element*'s custom element definition to *definition*.

    9. Set *element*'s `is` value to *isValue*.

    10. Return *element*.

    Note

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

10. Let *prototype* be ? Get(NewTarget, "prototype").

11. If *prototype* is not an Object:

    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.*

12. Let *element* be the last entry in *definition*'s construction stack[p798].

13. If *element* is an *already constructed* marker[p798], then throw a `TypeError`.

    Example

This can occur when the author code inside the custom element constructor[p796] non-conformantly[p794] creates another instance of the class being constructed, before calling `super()`:

```
let doSillyThing = true;

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 a TypeError:
    super();
  }
}
```

Example

This can also occur when author code inside the custom element constructor[p796] non-conformantly[p794] calls `super()` twice, since per the JavaScript specification, this actually executes the superclass constructor (i.e. this algorithm) twice, before throwing an error:

```
class DontDoThisEither extends HTMLElement {
  constructor() {
    super();

    // This will throw, but not until it has already called into the HTMLElement
constructor
    super();
  }
}
```

14. Perform ? *element*.[[SetPrototypeOf]](*prototype*).
15. Replace the last entry in *definition*'s construction stack[p798] with an *already constructed* marker[p798].
16. Return *element*.

Note

*This step is normally reached when upgrading[p803] a custom element; the existing element is returned, so that the `super()` call inside the custom element constructor[p796] assigns that existing element to **this**.*

In addition to the constructor behavior implied by `[HTMLConstructor]`[p151], 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`[p799] clause when defining a custom element constructor[p796]:

```
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 §[p154]

Each element in this specification has a definition that includes the following information:

Categories

A list of categories[p155] to which the element belongs. These are used when defining the content models[p155] 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[p157] is flow content[p157]. Thus, elements that are phrasing content[p157] will only be listed as "where phrasing content[p157] is expected", since this is the more-specific expectation. Anywhere that expects flow content[p157] also expects phrasing content[p157], 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`[p1546] syntax, the start[p1358] and end[p1359] tags can be omitted. This information is redundant with the normative requirements given in the optional tags[p1360] 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`[p70] and `aria-*`[p70] attributes are defined in *ARIA in HTML*. [ARIA][p1579] [ARIAHTML][p1579]

For implementers: User agent requirements for implementing accessibility API semantics are defined in *HTML Accessibility API Mappings*. [HTMLAAM][p1582]

Sanitization

The Sanitization[p154] information for each element defines its sanitization category[p1249], which affects how the element is processed during sanitization. It can also define one or more of the element's attributes as navigating URL attributes[p1251].

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[p148], along with any additional normative conformance criteria that may apply to authors and implementations. Examples are sometimes also included.

3.2.4.1 Attributes §[p154]

An attribute value is a string. Except where otherwise specified, attribute values on HTML elements[p46] 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 §p155

Each element defined in this specification has a content model: a description of the element's expected contentsp155. An HTML elementp46 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 whitespacep155, 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 whitespacep155) between them. Similarly, a node is the* only child *of an element if that element contains no other nodes other than inter-element whitespacep155, comment nodes, and processing instruction nodes.*

Authors must not use HTML elementsp46 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`p266 element. Thus, a `div`p266 element is allowed in that context, even though this is not explicitly normatively stated by this specification. [ATOM]p1579

In addition, HTML elementsp46 may be orphan nodes (i.e. without a parent node).

Example

For example, creating a `td`p512 element and storing it in a global variable in a script is conforming, even though `td`p512 elements are otherwise only supposed to be used inside `tr`p510 elements.

```
var data = {
  name: "Banana",
  cell: document.createElement('td'),
};
```

3.2.5.1 The "nothing" content model §p155

When an element's content model is **nothing**, the element must contain no `Text` nodes (other than inter-element whitespacep155) and no element nodes.

Note

Most HTML elements whose content model is "nothing" are also, for convenience, void elementsp1357 (elements that have no end tagp1359 in the HTML syntaxp1356). However, these are entirely separate concepts.

3.2.5.2 Kinds of content §p155

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 contentp156
- Flow contentp157
- Sectioning contentp157

- Heading content[p157]
- Phrasing content[p157]
- Embedded content[p158]
- Interactive content[p158]

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 §[p156]

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[p182], link[p184], meta[p196], noscript[p703], script[p685], style[p207], template[p705], title[p181]

Elements from other namespaces whose semantics are primarily metadata-related (e.g. RDF) are also metadata content[p156].

Example

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

```
<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/swap/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 string, 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 §p15

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

⇒ a, abbr, address, area (if it is a descendant of a map element), article, aside, audio, b, bdi, bdo, blockquote, br, button, canvas, cite, code, data, datalist, del, details, dfn, dialog, div, dl, em, embed, fieldset, figure, footer, form, h1, h2, h3, h4, h5, h6, header, hgroup, hr, i, iframe, img, input, ins, kbd, label, link (if it is allowed in the body), main (if it is a hierarchically correct main element), map, mark, MathML math, menu, meta (if the itemprop attribute is present), meter, nav, noscript, object, ol, output, p, picture, pre, progress, q, ruby, s, samp, script, search, section, select, slot, small, span, strong, sub, sup, SVG svg, table, template, textarea, time, u, ul, var, video, wbr, autonomous custom elements, text

3.2.5.2.3 Sectioning content §p15

Sectioning content is content that defines the scope of header and footer elements.

⇒ article, aside, nav, section

3.2.5.2.4 Heading content §p15

Heading content defines the heading 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 (if it has a descendant h1 to h6 element)

3.2.5.2.5 Phrasing content §p15

Phrasing content is the text of the document, as well as elements that mark up that text at the intra-paragraph level. Runs of phrasing content form paragraphs.

⇒ a, abbr, area (if it is a descendant of a map element), audio, b, bdi, bdo, br, button, canvas, cite, code, data, datalist, del, dfn, em, embed, i, iframe, img, input, ins, kbd, label, link (if it is allowed in the body), map, mark, MathML math, meta (if the itemprop attribute is present), meter, noscript, object, output, picture, progress, q, ruby, s, samp, script, select, selectedcontent (if it is a descendant of a button in a select), slot, small, span, strong, sub, sup, SVG 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[p158] is sometimes used as a content model on its own, but is also phrasing content[p157], and can be inter-element whitespace[p155] (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 §[p158]

Embedded content is content that imports another resource into the document, or content from another vocabulary that is inserted into the document.

> ⇒ audio[p425], canvas[p713], embed[p413], iframe[p404], img[p359], MathML math, object[p416], picture[p355], SVG svg, video[p420]

Elements that are from namespaces other than the HTML namespace and that convey content but not metadata, are embedded content[p158] for the purposes of the content models defined in this specification. (For example, MathML or SVG.)

Some embedded content elements can have **fallback content**: content that is to be used when the external resource cannot be used (e.g. because it is of an unsupported format). The element definitions state what the fallback is, if any.

3.2.5.2.7 Interactive content §[p158]

Interactive content is content that is specifically intended for user interaction.

> ⇒ a[p267] (if the href[p314] attribute is present), audio[p425] (if the controls[p481] attribute is present), button[p585], details[p666], embed[p413], iframe[p404], img[p359] (if the usemap[p491] or controls[p363] attribute is present), input[p539] (if the type[p541] attribute is *not* in the Hidden[p546] state), label[p536], select[p590], textarea[p606], video[p420] (if the controls[p481] attribute is present)

3.2.5.2.8 Palpable content §[p158]

As a general rule, elements whose content model allows any flow content[p157] or phrasing content[p157] should have at least one node in its contents[p155] that is **palpable content** and that does not have the hidden[p862] attribute specified.

Note

Palpable content[p158] makes an element non-empty by providing either some descendant non-empty text[p158], or else something users can hear (audio[p425] elements) or view (video[p420], img[p359], or canvas[p713] 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[p267], abbr[p279], address[p230], article[p214], aside[p222], audio[p425] (if the controls[p481] attribute is present), b[p303], bdi[p307], bdo[p309], blockquote[p244], button[p585], canvas[p713], cite[p275], code[p297], data[p289], del[p351], details[p666], dfn[p278], div[p266], dl[p253] (if the element's children include at least one name-value group), em[p270], embed[p413], fieldset[p620], figure[p259], footer[p227], form[p532], h1[p224], h2[p224], h3[p224], h4[p224], h5[p224], h6[p224], header[p226], hgroup[p225], i[p302], iframe[p404], img[p359], input[p539] (if the type[p541] attribute is *not* in the Hidden[p546] state), ins[p350], kbd[p300], label[p536], main[p262], map[p487], mark[p305], MathML math, menu[p250] (if the element's children include at least one li[p251] element), meter[p616], nav[p219], object[p416], ol[p247] (if the element's children include at least one li[p251] element), output[p611], p[p238], picture[p355], pre[p242], progress[p614], q[p277], ruby[p281], s[p274], samp[p299], search[p264], section[p216], select[p590], small[p272], span[p309], strong[p271], sub[p301], sup[p301], SVG svg, table[p496], textarea[p606], time[p290], u[p304], ul[p249] (if the element's children include at least one li[p251] element), var[p298], video[p420], autonomous custom elements[p796], text[p158] that is not inter-element whitespace[p155]

3.2.5.2.9 Script-supporting elements §[p15]

Script-supporting elements are those that do not represent[p148] 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[p685], template[p705]

3.2.5.3 Transparent content models §[p15]

Some elements are described as **transparent**; they have "transparent" in the description of their content model. The content model of a transparent[p159] 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[p350] element inside a ruby[p281] element cannot contain an rt[p287] element, because the part of the ruby[p281] element's content model that allows ins[p350] elements is the part that allows phrasing content[p157], and the rt[p287] element is not phrasing content[p157].

Note

In some cases, where transparent elements are nested in each other, the process has to be applied iteratively.

Example

Consider the following markup fragment:

```
<p><object><ins><map><a href="/">Apples</a></map></ins></object></p>
```

To check whether "Apples" is allowed inside the a[p267] element, the content models are examined. The a[p267] element's content model is transparent, as is the map[p487] element's, as is the ins[p350] element's, as is the object[p416] element's. The object[p416] element is found in the p[p238] element, whose content model is phrasing content[p157]. Thus, "Apples" is allowed, as text is phrasing content.

When a transparent element has no parent, then the part of its content model that is "transparent" must instead be treated as accepting any flow content[p157].

3.2.5.4 Paragraphs §[p15]

Note

The term paragraph[p159] as defined in this section is used for more than just the definition of the p[p238] element. The paragraph[p159] concept defined here is used to describe how to interpret documents. The p[p238] element is merely one of several ways of marking up a paragraph[p159].

A **paragraph** is typically a run of phrasing content[p157] that forms a block of text with one or more sentences that discuss a particular topic, as in typography, but can also be used for more general thematic grouping. For instance, an address is also a paragraph, as is a part of a form, a byline, or a stanza in a poem.

Example

In the following example, there are two paragraphs in a section. There is also a heading, which contains phrasing content that is not a paragraph. Note how the comments and inter-element whitespace[p155] do not form paragraphs.

```
<section>
  <h2>Example of paragraphs</h2>
  This is the <em>first</em> paragraph in this example.
```

```
  <p>This is the second.</p>
  <!-- This is not a paragraph. -->
</section>
```

Paragraphs in flow content[p157] are defined relative to what the document looks like without the a[p267], ins[p350], del[p351], and map[p487] elements complicating matters, since those elements, with their hybrid content models, can straddle paragraph boundaries, as shown in the first two examples below.

Note

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

Example

The following example takes the markup from the earlier example and puts ins[p350] and del[p351] 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[p350] and del[p351] elements — the ins[p350] element straddles the heading and the first paragraph, and the del[p351] element straddles the boundary between the two paragraphs.

```
<section>
  <ins><h2>Example of paragraphs</h2>
  This is the <em>first</em> paragraph in</ins> this example<del>.
  <p>This is the second.</p></del>
  <!-- This is not a paragraph. -->
</section>
```

Let *view* be a view of the DOM that replaces all a[p267], ins[p350], del[p351], and map[p487] elements in the document with their contents[p155]. Then, in *view*, for each run of sibling phrasing content[p157] nodes uninterrupted by other types of content, in an element that accepts content other than phrasing content[p157] as well as phrasing content[p157], 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[p158] nor inter-element whitespace[p155], a paragraph exists in the original DOM from immediately before *first* to immediately after *last*. (Paragraphs can thus span across a[p267], ins[p350], del[p351], and map[p487] elements.)

Conformance checkers may warn authors of cases where they have paragraphs that overlap each other (this can happen with object[p416], video[p420], audio[p425], and canvas[p713] elements, and indirectly through elements in other namespaces that allow HTML to be further embedded therein, like SVG svg or MathML math).

A paragraph[p159] is also formed explicitly by p[p238] elements.

Note

The p[p238] 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.

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

Here is another way of marking this up, this time showing the paragraphs explicitly, and splitting the one link element into three:

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

Example

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

```
<section>
 <h2>My Cats</h2>
 You can play with my cat simulator.
 <object data="cats.sim">
  To see the cat simulator, use one of the following links:
  <ul>
   <li><a href="cats.sim">Download simulator file</a>
   <li><a href="https://sims.example.com/watch?v=LYds5xY4INU">Use online simulator</a>
  </ul>
  Alternatively, upgrade to the Mellblom Browser.
 </object>
 I'm quite proud of it.
</section>
```

There are five paragraphs:

1. The paragraph that says "You can play with my cat simulator. *object* I'm quite proud of it.", where *object* is the object[p416] element.
2. The paragraph that says "To see the cat simulator, use one of the following links:".
3. The paragraph that says "Download simulator file".
4. The paragraph that says "Use online simulator".
5. The paragraph that says "Alternatively, upgrade to the Mellblom Browser.".

The first paragraph is overlapped by the other four. A user agent that supports the "cats.sim" resource will only show the first one, but a user agent that shows the fallback will confusingly show the first sentence of the first paragraph as if it was in the same paragraph as the second one, and will show the last paragraph as if it was at the start of the second sentence of the first paragraph.

To avoid this confusion, explicit p[p238] elements can be used. For example:

```
<section>
 <h2>My Cats</h2>
 <p>You can play with my cat simulator.</p>
 <object data="cats.sim">
  <p>To see the cat simulator, use one of the following links:</p>
  <ul>
   <li><a href="cats.sim">Download simulator file</a>
   <li><a href="https://sims.example.com/watch?v=LYds5xY4INU">Use online simulator</a>
  </ul>
  <p>Alternatively, upgrade to the Mellblom Browser.</p>
 </object>
 <p>I'm quite proud of it.</p>
</section>
```

3.2.6 Global attributes §[p162]

The following attributes are common to and may be specified on all HTML elements[p46] (even those not defined in this specification):

- accesskey[p890]
- autocapitalize[p898]
- autocorrect[p899]
- autofocus[p887]
- contenteditable[p892]
- dir[p167]
- draggable[p924]
- enterkeyhint[p901]
- headingoffset[p232]
- headingreset[p232]
- hidden[p862]
- inert[p866]
- inputmode[p900]
- is[p796]
- itemid[p832]
- itemprop[p833]
- itemref[p832]
- itemscope[p831]
- itemtype[p831]
- lang[p165]
- nonce[p104]
- popover[p925]
- spellcheck[p895]
- style[p171]
- tabindex[p878]
- title[p164]
- translate[p166]
- writingsuggestions[p896]

These attributes are only defined by this specification as attributes for HTML elements[p46]. When this specification refers to elements having these attributes, elements from namespaces that are not defined as having these attributes must not be considered as being elements with these attributes.

Example

For example, in the following XML fragment, the "bogus" element does not have a dir[p167] attribute as defined in this specification, despite having an attribute with the literal name "dir". Thus, the directionality[p168] of the inner-most span[p309] element is 'rtl[p168]', inherited from the div[p266] element indirectly through the "bogus" element.

```
<div xmlns="http://www.w3.org/1999/xhtml" dir="rtl">
 <bogus xmlns="https://example.net/ns" dir="ltr">
  <span xmlns="http://www.w3.org/1999/xhtml">
  </span>
 </bogus>
</div>
```

DOM defines the user agent requirements for the **class**, **id**, and **slot** attributes for any element in any namespace. [DOM][p1582]

The class[p162], id[p162], and slot[p162] attributes may be specified on all HTML elements[p46].

When specified on HTML elements[p46], the class[p162] attribute must have a value that is a set of space-separated tokens[p99] 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.

There are no additional restrictions on the tokens authors can use in the class[p162] attribute, but authors are encouraged to use values that describe the nature of the content, rather than values that describe the desired presentation of the content.

When specified on HTML elements[p46], the id[p162] attribute value must be unique amongst all the IDs in the element's tree and must contain at least one character. The value must not contain any ASCII whitespace.

Note

The id[p162] 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[p162] attribute.

There are no conformance requirements for the slot[p162] attribute specific to HTML elements[p46].

Note

The slot[p162] attribute is used to assign a slot to an element: an element with a slot[p162] attribute is assigned to the slot created by the slot[p712] element whose name[p712] attribute's value matches that slot[p162] attribute's value — but only if that slot[p712] element finds itself in the shadow tree whose root's host has the corresponding slot[p162] 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[p178] can be specified (the ARIA role[p70] and aria-*[p70] attributes). [ARIA][p1579]

The following event handler content attributes[p1207] may be specified on any HTML element[p46]:

- onauxclick[p1213]
- onbeforeinput[p1213]
- onbeforematch[p1213]
- onbeforetoggle[p1213]
- onblur[p1215]*
- oncancel[p1213]
- oncanplay[p1213]
- oncanplaythrough[p1213]
- onchange[p1213]
- onclick[p1213]
- onclose[p1213]
- oncommand[p1213]
- oncontextlost[p1213]
- oncontextmenu[p1214]
- oncontextrestored[p1214]
- oncopy[p1214]
- oncuechange[p1214]
- oncut[p1214]
- ondblclick[p1214]
- ondrag[p1214]
- ondragend[p1214]
- ondragenter[p1214]
- ondragleave[p1214]
- ondragover[p1214]
- ondragstart[p1214]
- ondrop[p1214]
- ondurationchange[p1214]
- onemptied[p1214]
- onended[p1214]
- onerror[p1215]*
- onfocus[p1215]*
- onformdata[p1214]
- oninput[p1214]
- oninvalid[p1214]
- onkeydown[p1214]
- onkeypress[p1214]
- onkeyup[p1214]
- onload[p1215]*
- onloadeddata[p1214]
- onloadedmetadata[p1214]
- onloadstart[p1214]
- onmousedown[p1214]
- onmouseenter[p1214]

- onmouseleave[p1214]
- onmousemove[p1214]
- onmouseout[p1214]
- onmouseover[p1214]
- onmouseup[p1214]
- onpaste[p1214]
- onpause[p1214]
- onplay[p1214]
- onplaying[p1214]
- onprogress[p1214]
- onratechange[p1214]
- onreset[p1214]
- onresize[p1215]*
- onscroll[p1215]*
- onscrollend[p1214]*
- onsecuritypolicyviolation[p1214]
- onseeked[p1214]
- onseeking[p1214]
- onselect[p1214]
- onslotchange[p1214]
- onstalled[p1214]
- onsubmit[p1214]
- onsuspend[p1214]
- ontimeupdate[p1214]
- ontoggle[p1214]
- onvolumechange[p1214]
- onwaiting[p1214]
- onwheel[p1215]

Note

The attributes marked with an asterisk have a different meaning when specified on body[p212] elements as those elements expose event handlers[p1206] of the Window[p965] object with the same names.

Note

While these attributes apply to all elements, they are not useful on all elements. For example, only media elements[p430] will ever receive a volumechange[p486] event fired by the user agent.

Custom data attributes[p171] (e.g. `data-foldername` or `data-msgid`) can be specified on any HTML element[p46], to store custom data, state, annotations, and similar, specific to the page.

In HTML documents, elements in the HTML namespace may have an `xmlns` attribute specified, if, and only if, it has the exact value "`http://www.w3.org/1999/xhtml`". This does not apply to XML documents.

Note

In HTML, the `xmlns` attribute has absolutely no effect. It is basically a talisman. It is allowed merely to make migration to and from XML mildly easier. When parsed by an HTML parser[p1368], the attribute ends up in no namespace. In XML, the attribute is part of the namespace declaration mechanism and always ends up in the "`http://www.w3.org/2000/xmlns/`" namespace.

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[p46], as the default behavior in HTML is to preserve whitespace. [XML][p1588]

Note

There is no way to serialize the xml:space attribute on HTML elements[p46] in the text/html[p1546] syntax.

3.2.6.1 The title[p164] attribute §[p164]

The **title** attribute represents[p148] 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[p158], it could be a label for, or instructions for, use of the element; and so forth. The value is text.

Note

Relying on the `title`[p164] 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`[p164] attribute of the nearest ancestor HTML element[p46] with a `title`[p164] 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`[p164] attribute's value contains U+000A LINE FEED (LF) characters, the content is split into multiple lines. Each U+000A LINE FEED (LF) character represents a line break.

Example

Caution is advised with respect to the use of newlines in `title`[p164] 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`[p184], `abbr`[p279], and `input`[p539], define additional semantics for the `title`[p164] 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`[p164] attribute, then return the result of running normalize newlines on its value.
2. If the element has a parent element, then return the parent element's advisory information[p165].
3. Return the empty string.

User agents should inform the user when elements have advisory information[p165], otherwise the information would not be discoverable.

3.2.6.2 The `lang`[p165] and `xml:lang` attributes §[p165]

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][p1579]

The **`lang`** attribute in the XML namespace is defined in XML. [XML][p1588]

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 (except for `slot`[p712] elements in a shadow tree).

The `lang`[p165] attribute in no namespace may be used on any HTML element[p46].

The lang attribute in the XML namespace may be used on HTML elements[p46] 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`[p165] 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[p46] 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[p46] in HTML documents, but such attributes must only be specified if a `lang`[p165] 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 use the first appropriate step in the following list:

↪ **If the node is an element that has a lang attribute in the XML namespace set**

Use the value of that attribute.

↪ **If the node is an HTML element[p46] or an element in the SVG namespace, and it has a lang[p165] in no namespace attribute set**

Use the value of that attribute.

↪ **If the node's parent is a shadow root**

Use the language[p166] of that shadow root's host.

↪ **If the node's parent element is not null**

Use the language[p166] of that parent element.

↪ **Otherwise**

If there is a pragma-set default language[p203] set, then that is the language of the node. If there is no pragma-set default language[p203] 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 roundtripping 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][p1579]

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(abcde)`, even though both are equally invalid. Similarly, if a web browser and screen reader working in unison communicated about the language of the element, the browser would tell the screen reader that the language was "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).

3.2.6.3 The translate[p166] attribute §[p166]

The **translate** attribute 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. It is an enumerated attribute[p80] with the following keywords and states:

Keyword	State	Brief description
yes	Yes	Sets translation mode[p167] to translate-enabled[p167].
no	No	Sets translation mode[p167] to no-translate[p167].

The attribute's *missing value default*[p80] and *invalid value default*[p80] are both the **Inherit** state, and its *empty value default*[p80] is the Yes[p166] state.

Each element (even non-HTML elements) has a **translation mode**, which is in either the translate-enabled[p167] state or the no-translate[p167] state. If an HTML element[p46]'s `translate`[p166] attribute is in the Yes[p166] state, then the element's translation mode[p167] is in the translate-enabled[p167] state; otherwise, if the element's `translate`[p166] attribute is in the No[p166] state, then the element's translation mode[p167] is in the no-translate[p167] state. Otherwise, either the element's `translate`[p166] attribute is in the Inherit[p166] state, or the element is not an HTML element[p46] and thus does not have a `translate`[p166] attribute; in either case, the element's translation mode[p167] is in the same state as its parent element's, if any, or in the translate-enabled[p167] state, if the element's parent element is null.

When an element is in the **translate-enabled** state, the element's translatable attributes[p167] 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`[p514] on `th`[p513] elements
- `alt` on `area`[p490], `img`[p360], and `input`[p567] elements
- `content`[p197] on `meta`[p196] elements, if the `name`[p197] attribute specifies a metadata name whose value is known to be translatable
- `download`[p314] on `a`[p267] and `area`[p489] elements
- `label` on `optgroup`[p601], `option`[p603], and `track`[p429] elements
- `lang`[p165] on HTML elements[p46]; must be "translated" to match the language used in the translation
- `placeholder` on `input`[p578] and `textarea`[p610] elements
- `srcdoc`[p405] on `iframe`[p404] elements; must be parsed and recursively processed
- `style`[p171] on HTML elements[p46]; must be parsed and recursively processed (e.g. for the values of 'content' properties)
- `title`[p164] on all HTML elements[p46]
- `value`[p543] on `input`[p539] elements with a `type`[p541] attribute in the Button[p569] state or the Reset Button[p568] state

Other specifications may define other attributes that are also translatable attributes[p167]. 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[p167] is translate-enabled[p167], 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:

```
<!DOCTYPE HTML>
<html lang=en> <!-- default on the document element is translate=yes -->
 <head>
  <title>The Bee Game</title> <!-- implied translate=yes inherited from ancestors -->
 </head>
 <body>
  <p>The Bee Game is a text adventure game in English.</p>
  <p>When the game launches, the first thing you should do is type
  <kbd translate=no>eat honey</kbd>. The game will respond with:</p>
  <pre><samp translate=no>Yum yum! That was some good honey!</samp></pre>
 </body>
</html>
```

3.2.6.4 The `dir`[p167] attribute §[p167]

The **`dir`** attribute is an enumerated attribute[p80] with the following keywords and states:

Keyword	State	Brief description
`ltr`	**LTR**	The contents of the element are explicitly directionally isolated left-to-right text.
`rtl`	**RTL**	The contents of the element are explicitly directionally isolated right-to-left text.
`auto`	**Auto**	The contents of the element are explicitly directionally isolated text, but the direction is to be determined programmatically using the contents

Keyword	State	Brief description
		of the element (as described below).

Note

The heuristic used by the Auto[p167] 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][p1579]

For textarea[p606] and pre[p242] elements, the heuristic is applied on a per-paragraph level.

The attribute's *missing value default*[p80] and *invalid value default*[p80] are both the **Undefined** state.

The **directionality** of an element (any element, not just an HTML element[p46]) is either '**ltr**' or '**rtl**'. To compute the directionality[p168] given an element *element*, switch on *element*'s dir[p167] attribute state:

↪ **LTR**[p167]
Return 'ltr[p168]'.

↪ **RTL**[p167]
Return 'rtl[p168]'.

↪ **Auto**[p167]

1. Let *result* be the auto directionality[p168] of *element*.
2. If *result* is null, then return 'ltr[p168]'.
3. Return *result*.

↪ **Undefined**[p168]

↪ **If *element* is a bdi[p307] element**

1. Let *result* be the auto directionality[p168] of *element*.
2. If *result* is null, then return 'ltr[p168]'.
3. Return *result*.

↪ **If *element* is an input[p539] element whose type[p541] attribute is in the Telephone[p547] state**
Return 'ltr[p168]'.

↪ **Otherwise**
Return the parent directionality[p169] of *element*.

Note

Since the dir[p167] attribute is only defined for HTML elements[p46], it cannot be present on elements from other namespaces. Thus, elements from other namespaces always end up using the parent directionality[p169].

The **auto-directionality form-associated elements** are:

- input[p539] elements whose type[p541] attribute is in the Hidden[p546], Text[p546], Search[p546], Telephone[p547], URL[p547], Email[p548], Password[p550], Submit Button[p565], Reset Button[p568], or Button[p569] state, and
- textarea[p606] elements.

To compute the **auto directionality** given an element *element*:

1. If *element* is an auto-directionality form-associated element[p168]:
   1. If *element*'s value[p626] contains a character of bidirectional character type AL or R, and there is no character of bidirectional character type L anywhere before it in the element's value[p626], then return 'rtl[p168]'. [BIDI][p1579]
   2. If *element*'s value[p626] is not the empty string, then return 'ltr[p168]'.

3. Return null.

2. If *element* is a `slot`[p712] element whose root is a shadow root and *element*'s assigned nodes are not empty:

    1. For each node *child* of *element*'s assigned nodes:

        1. Let *childDirection* be null.

        2. If *child* is a `Text` node, then set *childDirection* to the text node directionality[p169] of *child*.

        3. Otherwise:

            1. Assert: *child* is an `Element` node.

            2. Set *childDirection* to the contained text auto directionality[p169] of *child* with *canExcludeRoot*[p169] set to true.

        4. If *childDirection* is not null, then return *childDirection*.

    2. Return null.

3. Return the contained text auto directionality[p169] of *element* with *canExcludeRoot*[p169] set to false.

To compute the **contained text auto directionality** of an element *element* with a boolean ***canExcludeRoot***:

1. For each node *descendant* of *element*'s descendants, in tree order:

    1. If any of

        - *descendant*
        - any ancestor element of *descendant* that is a descendant of *element*
        - if *canExcludeRoot* is true, *element*

        is one of

        - a `bdi`[p307] element
        - a `script`[p685] element
        - a `style`[p207] element
        - a `textarea`[p606] element
        - an element whose `dir`[p167] attribute is not in the Undefined[p168] state

        then continue.

    2. If *descendant* is a `slot`[p712] element whose root is a shadow root, then return the directionality[p168] of that shadow root's host.

    3. If *descendant* is not a `Text` node, then continue.

    4. Let *result* be the text node directionality[p169] of *descendant*.

    5. If *result* is not null, then return *result*.

2. Return null.

To compute the **text node directionality** given a `Text` node *text*:

1. If *text*'s data does not contain a code point whose bidirectional character type is L, AL, or R, then return null. [BIDI][p1579]

2. Let *codePoint* be the first code point in *text*'s data whose bidirectional character type is L, AL, or R.

3. If *codePoint* is of bidirectional character type AL or R, then return '`rtl`[p168]'.

4. If *codePoint* is of bidirectional character type L, then return '`ltr`[p168]'.

To compute the **parent directionality** given an element *element*:

1. Let *parentNode* be *element*'s parent node.

2. If *parentNode* is a shadow root, then return the directionality[p168] of *parentNode*'s host.

3. If *parentNode* is an element, then return the directionality[p168] of *parentNode*.

4. Return '`ltr`[p168]'.

Note

This attribute has rendering requirements involving the bidirectional algorithm[p177].

The **directionality of an attribute** of an HTML element[p46], which is used when the text of that attribute is to be included in the rendering in some manner, is determined as per the first appropriate set of steps from the following list:

↪ **If the attribute is a directionality-capable attribute[p170] and the element's dir[p167] attribute is in the Auto[p167] state**

Find the first character (in logical order) of the attribute's value that is of bidirectional character type L, AL, or R. [BIDI][p1579]

If such a character is found and it is of bidirectional character type AL or R, the directionality of the attribute[p170] is '`rtl`[p168]'.

Otherwise, the directionality of the attribute[p170] is '`ltr`[p168]'.

↪ **Otherwise**

The directionality of the attribute[p170] is the same as the element's directionality[p168].

The following attributes are **directionality-capable attributes**:

- `abbr`[p514] on `th`[p513] elements
- `alt` on `area`[p490], `img`[p360], and `input`[p567] elements
- `content`[p197] on `meta`[p196] elements, if the `name`[p197] attribute specifies a metadata name whose value is primarily intended to be human-readable rather than machine-readable
- `label` on `optgroup`[p601], `option`[p603], and `track`[p429] elements
- `placeholder` on `input`[p578] and `textarea`[p610] elements
- `title`[p164] on all HTML elements[p46]

For web developers (non-normative)

***document*.dir[p170] [= *value*]**

Returns the `html` element[p142]'s `dir`[p167] attribute's value, if any.

Can be set, to either "`ltr`", "`rtl`", or "`auto`" to replace the `html` element[p142]'s `dir`[p167] attribute's value.

If there is no `html` element[p142], returns the empty string and ignores new values.

The **`dir`** IDL attribute on an element must reflect[p108] the `dir`[p167] content attribute of that element, limited to only known values[p109].

The **`dir`** IDL attribute on `Document`[p135] objects must reflect[p108] the `dir`[p167] content attribute of the `html` element[p142], if any, limited to only known values[p109]. If there is no such element, then the attribute must return the empty string and do nothing on setting.

Note

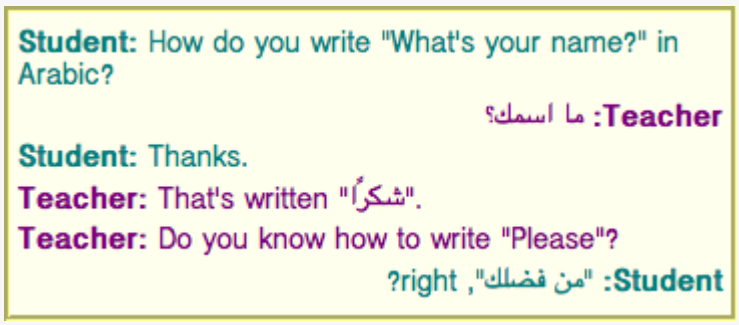
Authors are strongly encouraged to use the `dir`[p167] attribute to indicate text direction rather than using CSS, since that way their documents will continue to render correctly even in the absence of CSS (e.g. as interpreted by search engines).

Example

This markup fragment is of an IM conversation.

```
<p dir=auto class="u1"><b><bdi>Student</bdi>:</b> How do you write "What's your name?" in
Arabic?</p>
<p dir=auto class="u2"><b><bdi>Teacher</bdi>:</b> ما اسمك؟</p>
<p dir=auto class="u1"><b><bdi>Student</bdi>:</b> Thanks.</p>
<p dir=auto class="u2"><b><bdi>Teacher</bdi>:</b> That's written "شكرًا".</p>
<p dir=auto class="u2"><b><bdi>Teacher</bdi>:</b> Do you know how to write "Please"?</p>
<p dir=auto class="u1"><b><bdi>Student</bdi>:</b> "من فضلك", right?</p>
```

Given a suitable style sheet and the default alignment styles for the `p`[p238] element, namely to align the text to the *start edge* of the paragraph, the resulting rendering could be as follows:



As noted earlier, the `auto`[p167] 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`[p171] attribute §[p17 1]

All HTML elements[p46] may have the **`style`** content attribute set. This is a style attribute as defined by *CSS Style Attributes*. [CSSATTR][p1580]

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. [CSSATTR][p1580]

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 not be applied to the element. [CSP][p1580]

Documents that use `style`[p171] attributes on any of their elements must still be comprehensible and usable if those attributes were removed.

Note

In particular, using the `style`[p171] 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`[p862] attribute.)

For web developers (non-normative)

***element*.style**

Returns a `CSSStyleDeclaration` object for the element's `style`[p171] attribute.

The `style` IDL attribute is defined in *CSS Object Model*. [CSSOM][p1581]

Example

In the following example, the words that refer to colors are marked up using the `span`[p309] element and the `style`[p171] attribute to make those words show up in the relevant colors in visual media.

```
<p>My sweat suit is <span style="color: green; background:
transparent">green</span> and my eyes are <span style="color: blue;
background: transparent">blue</span>.</p>
```

3.2.6.6 Embedding custom non-visible data with the `data-*`[p171] attributes §[p17 1]

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 a valid attribute local name, and contains no ASCII upper alphas.

Note

All attribute names on HTML elements[p46] in HTML documents get ASCII-lowercased automatically, so the restriction on ASCII uppercase letters doesn't affect such documents.

Custom data attributes[p171] 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[p826] 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.

```
<ol>
 <li data-length="2m11s">Beyond The Sea</li>
 ...
</ol>
```

It would be inappropriate, however, for the user to use generic software not associated with that music site to search for tracks of a certain length by looking at this data.

This is because these attributes are intended for use by the site's own scripts, and are not a generic extension mechanism for publicly-usable metadata.

Example

Similarly, a page author could write markup that provides information for a translation tool that they are intending to use:

```
<p>The third <span data-mytrans-de="Anspruch">claim</span> covers the case of <span
translate="no">HTML</span> markup.</p>
```

In this example, the "`data-mytrans-de`" attribute gives specific text for the MyTrans product to use when translating the phrase "claim" to German. However, the standard `translate`[p166] attribute is used to tell it that in all languages, "HTML" is to remain unchanged. When a standard attribute is available, there is no need for a custom data attribute[p171] to be used.

Example

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.

```
<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[p79]; 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.

Every HTML element[p46] may have any number of custom data attributes[p171] 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[p171], as they are considered to be part of the page on which they are used. Authors of libraries that are reused by many authors are encouraged to include their name in the attribute names, to reduce the risk of clashes. Where it makes sense, library authors are also encouraged to make the exact name used in the attribute names customizable, so that libraries whose authors unknowingly picked the same name can be used on the same page, and so that multiple versions of a particular library can be used on the same page even when those versions are not mutually compatible.

Example

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`).

For web developers (non-normative)

***element*.dataset**[p173]

Returns a DOMStringMap[p173] object for the element's data-*[p171] attributes.

Hyphenated names become camel-cased. For example, `data-foo-bar=""` becomes `element.dataset.fooBar`.

The **dataset** IDL attribute provides convenient accessors for all the data-*[p171] attributes on an element. On getting, the dataset[p173] IDL attribute must return a DOMStringMap[p173] whose associated element is this element.

The DOMStringMap[p173] interface is used for the dataset[p173] attribute. Each DOMStringMap[p173] has an **associated element**.

IDL

```
[Exposed=Window,
 LegacyOverrideBuiltIns]
interface DOMStringMap {
  getter DOMString (DOMString name);
  [CEReactions] setter undefined (DOMString name, DOMString value);
  [CEReactions] deleter undefined (DOMString name);
};
```

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[p173]'s associated element[p173] 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[p173] object at any instant are the names of each pair returned from getting the DOMStringMap's name-value pairs[p173] at that instant, in the order returned.

To determine the value of a named property *name* for a DOMStringMap[p173], 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[p173].

To set the value of a new named property or set the value of an existing named property for a DOMStringMap[p173], given a property name *name* and a new value *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* is not a valid attribute local name, then throw an "InvalidCharacterError" DOMException.

5. Set an attribute value for the DOMStringMap[p173]'s associated element[p173] using *name* and *value*.

To delete an existing named property *name* for a DOMStringMap[p173], 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[p173]'s associated element[p173].

Note

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[p173]. [WEBIDL][p1588]*

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[p162] attribute along with data-*[p171] 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>
</div>
```

Notice how the hyphenated attribute name becomes camel-cased in the API.

Example

Given the following fragment and elements with similar constructions:

```
<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:

```
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[p175] and outerText[p175] properties §[p174]

For web developers (non-normative)

***element*.innerText[p175] [= *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`[p310] elements.

***element*.outerText[p175] [= *value*]**

Returns the element's text content "as rendered".

Can be set, to replace the element with the given value, but with line breaks converted to `br`[p310] elements.

The **get the text steps**, given an HTMLElement[p149] *element*, are:

1. If *element* is not being rendered[p1488] or if the user agent is a non-CSS user agent, then return *element*'s descendant text content.

   Note

   This step can produce surprising results, as when the `innerText`[p175] getter is invoked on an element not being rendered[p1488], its text contents are returned, but when accessed on an element that is being rendered[p1488], all of its children that are not being rendered[p1488] have their text contents ignored.

2. Let *results* be a new empty list.
3. For each child node *node* of *element*:
   1. Let *current* be the list resulting from running the rendered text collection steps[p175] 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. Remove any runs of consecutive *required line break count* items at the start or end of *results*.
6. Replace each remaining run of consecutive *required line break count* items with a string consisting of as many U+000A LF code points as the maximum of the values in the *required line break count* items.
7. Return the concatenation of the string items in *results*.

The **`innerText`** and **`outerText`** getter steps are to return the result of running get the text steps[p175] with this.

✓ MDN

The **rendered text collection steps**, given a node *node*, are as follows:

1. Let *items* be the result of running the rendered text collection steps[p175] 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[p1488], 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`[p590] elements have an associated non-replaced inline CSS box whose child boxes include only those of `optgroup`[p600] and `option`[p601] element descendant nodes;
   - `optgroup`[p600] elements have an associated non-replaced block-level CSS box whose child boxes include only those of `option`[p601] element descendant nodes; and
   - `option`[p601] elements 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[p310] element. Soft hyphens should be preserved. [CSSTEXT][p1581]

5. If *node* is a br[p310] element, then append a string containing a single U+000A LF code point 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 TAB code point 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 LF code point to *items*.

8. If *node* is a p[p238] 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][p1580]

   Note

   Floats and absolutely-positioned elements fall into this category.

10. Return *items*.

Note

Note that descendant nodes of most replaced elements (e.g., textarea[p606], input[p539], and video[p420] — but not button[p585]) are not rendered by CSS, strictly speaking, and therefore have no CSS boxes for the purposes of this algorithm.

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 **set the inner text steps**, given an HTMLElement[p149] *element*, and a string *value* are:

1. Let *fragment* be the rendered text fragment[p176] for *value* given *element*'s node document.
2. Replace all with *fragment* within *element*.

The innerText[p175] setter steps are to run set the inner text steps[p176] with this and the given value.

The outerText[p175] setter steps are:

1. If this's parent is null, then throw a "NoModificationAllowedError" DOMException.
2. Let *next* be this's next sibling.
3. Let *previous* be this's previous sibling.
4. Let *fragment* be the rendered text fragment[p176] for the given value given this's node document.
5. If *fragment* has no children, then append a new Text node whose data is the empty string and node document is this's node document to *fragment*.
6. Replace this with *fragment* within this's parent.
7. If *next* is non-null and *next*'s previous sibling is a Text node, then merge with the next text node[p177] given *next*'s previous sibling.
8. If *previous* is a Text node, then merge with the next text node[p177] given *previous*.

The **rendered text fragment** for a string *input* given a Document[p135] *document* is the result of running the following steps:

1. Let *fragment* be the result of creating a document fragment given *document*.
2. Let *position* be a position variable for *input*, initially pointing at the start of *input*.

3. Let *text* be the empty string.
4. While *position* is not past the end of *input*:
    1. Collect a sequence of code points that are not U+000A LF or U+000D CR from *input* given *position*, and set *text* to the result.
    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 code point at *position* is either U+000A LF or U+000D CR:
        1. If the code point at *position* is U+000D CR and the next code point is U+000A LF, then advance *position* to the next code point in *input*.
        2. Advance *position* to the next code point in *input*.
        3. Append the result of creating an element given *document*, "br", and the HTML namespace to *fragment*.
5. Return *fragment*.

To **merge with the next text node** given a Text node *node*:

1. Let *next* be *node*'s next sibling.
2. If *next* is not a Text node, then return.
3. Replace data with *node*, *node*'s data's length, 0, and *next*'s data.
4. Remove *next*.

3.2.8 Requirements relating to the bidirectional algorithm §p17 7

3.2.8.1 Authoring conformance criteria for bidirectional-algorithm formatting characters §p17 7

Text contentp158 in HTML elementsp46 with Text nodes in their contentsp155, and text in attributes of HTML elementsp46 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). [BIDI]p1579

Note

Authors are encouraged to use the dirp167 attribute, the bdop309 element, and the bdip307 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 §p17 7

User agents must implement the Unicode bidirectional algorithm to determine the proper ordering of characters when rendering documents and parts of documents. [BIDI]p1579

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 renderingp1488 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. [CSSGC]p1581

The following elements and attributes have requirements defined by the renderingp1488 section that, due to the requirements in this section, are requirements on all user agents (not just those that support the suggested default renderingp49):

- dirp167 attribute
- bdip307 element
- bdop309 element
- brp310 element
- prep242 element
- textareap606 element

- wbr p311 element

3.2.9 Requirements related to ARIA and to platform accessibility APIs §p178

User agent requirements for implementing Accessibility API semantics on HTML elements p46 are defined in *HTML Accessibility API Mappings*. In addition to the rules there, for a custom element p796 *element*, the default ARIA role semantics are determined as follows: [HTMLAAM] p1582

1. Let *map* be *element*'s internal content attribute map p813.
2. If *map*["role"] exists, then return it.
3. Return no role.

Similarly, for a custom element p796 *element*, the default ARIA state and property semantics, for a state or property named *stateOrProperty*, are determined as follows:

1. If *element*'s attached internals p809 is non-null:
    1. If *element*'s attached internals p809's get the *stateOrProperty*-associated element p112 exists, then return the result of running it.
    2. If *element*'s attached internals p809's get the *stateOrProperty*-associated elements p113 exists, then return the result of running it.
2. If *element*'s internal content attribute map p813[*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]* p1579

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 p267 *element change as the* href p314 *attribute is added or removed.*

For an example of this in action, see the custom elements section p787.

Conformance checker requirements for checking use of ARIA role p70 and aria-* p70 attributes on HTML elements p46 are defined in *ARIA in HTML*. [ARIAHTML] p1579

4 The elements of HTML §p17 9

4.1 The document element §p17 9

4.1.1 The `html` element §p17 9

Categoriesp154**:**

None.

Contexts in which this element can be usedp154**:**

As document's document element.
Wherever a subdocument fragment is allowed in a compound document.

Content modelp154**:**

A `head`p180 element followed by a `body`p212 element.

Tag omission in text/htmlp154**:**

An `html`p179 element's start tagp1358 can be omitted if the first thing inside the `html`p179 element is not a commentp1367.
An `html`p179 element's end tagp1359 can be omitted if the `html`p179 element is not immediately followed by a commentp1367.

Content attributesp154**:**

Global attributesp162

Accessibility considerationsp154**:**

For authors.
For implementers.

Sanitizationp154**:**

Defaultp1249.

DOM interfacep154**:**

IDL

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

  // also has obsolete members
};
```

The `html`p179 element representsp148 the root of an HTML document.

Authors are encouraged to specify a `lang`p165 attribute on the root `html`p179 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`p179 element in the following example declares that the document's language is English.

```
<!DOCTYPE html>
<html lang="en">
<head>
<title>Swapping Songs</title>
</head>
<body>
<h1>Swapping Songs</h1>
<p>Tonight I swapped some of the songs I wrote with some friends, who
gave me some of the songs they wrote. I love sharing my music.</p>
</body>
</html>
```

4.2 Document metadata §[p180]

4.2.1 The head element §[p180]

Categories[p154]:

None.

Contexts in which this element can be used[p154]:

As the first element in an html[p179] element.

Content model[p154]:

If the document is an iframe srcdoc document[p405] or if title information is available from a higher-level protocol: Zero or more elements of metadata content[p156], of which no more than one is a title[p181] element and no more than one is a base[p182] element.

Otherwise: One or more elements of metadata content[p156], of which exactly one is a title[p181] element and no more than one is a base[p182] element.

Tag omission in text/html[p154]:

A head[p180] element's start tag[p1358] can be omitted if the element is empty, or if the first thing inside the head[p180] element is an element.

A head[p180] element's end tag[p1359] can be omitted if the head[p180] element is not immediately followed by ASCII whitespace or a comment[p1367].

Content attributes[p154]:

Global attributes[p162]

Accessibility considerations[p154]:

For authors.

For implementers.

Sanitization[p154]:

Default[p1249].

DOM interface[p154]:

IDL

```
[Exposed=Window]
interface HTMLHeadElement : HTMLElement {
  [HTMLConstructor] constructor();
};
```

The head[p180] element represents[p148] a collection of metadata for the Document[p135].

Example

The collection of metadata in a head[p180] element can be large or small. Here is an example of a very short one:

```
<!doctype html>
<html lang=en>
 <head>
  <title>A document with a short head</title>
 </head>
 <body>
 ...
```

Here is an example of a longer one:

```
<!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="STYLESHEET" HREF="default.css">
  <LINK REL="STYLESHEET ALTERNATE" HREF="big.css" TITLE="Big Text">
```

```
 <SCRIPT SRC="support.js"></SCRIPT>
 <META NAME="APPLICATION-NAME" CONTENT="Long headed application">
</HEAD>
<BODY>
...
```

Note

The title element is a required child in most situations, but when a higher-level protocol provides title information, e.g., in the subject line of an 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.

Sanitization:

Default.

DOM interface:

IDL

```
[Exposed=Window]
interface HTMLTitleElement : 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[p181] element's child text content.

The text[p182] attribute's setter must string replace all with the given value within this title[p181] element.

Example

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[p143] 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[p181] element are used in this way, the directionality[p168] of that title[p181] element should be used to set the directionality of the document's title in the user interface.

4.2.3 The base element §[p182]

Categories[p154]:

Metadata content[p156].

Contexts in which this element can be used[p154]:

In a head[p180] element containing no other base[p182] elements.

Content model[p154]:

Nothing[p155].

Tag omission in text/html[p154]:

No end tag[p1359].

Content attributes[p154]:

Global attributes[p162]
href[p183] — Document base URL[p101]
target[p183] — Default navigable[p1035] for hyperlink[p313] navigation[p1063] and form submission[p657]

Accessibility considerations[p154]:

For authors.
For implementers.

Sanitization[p154]:

Unsafe[p1249].

DOM interface[p154]:

IDL

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

  [CEReactions, ReflectSetter] attribute USVString href;
  [CEReactions, Reflect] attribute DOMString target;
```

```
};
```

The base element allows authors to specify the document base URL for the purposes of parsing URLs, and the name of the default navigable 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.

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 navigable target name or keyword, which specifies which navigable 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*, and an optional string-or-null *target* (default null), run these steps:

1. If *target* is null:
    1. If *element* has a `target` attribute, then set *target* to that attribute's value.
    2. Otherwise, if *element*'s node document contains a base element with a target attribute, set *target* to the value of the target attribute of the first such base element.
2. If *target* is not null, and contains an ASCII tab or newline and a U+003C (<), then set *target* to "`_blank`".
3. Return *target*.

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 content attribute with *document*'s fallback base URL, and *document*'s character encoding. (Thus, the base element isn't affected by itself.)
3. If any of the following are true:
    - *urlRecord* is failure;
    - *urlRecord*'s scheme is "`data`" or "`javascript`"; or
    - running Is base allowed for Document? on *urlRecord* and *document* returns "`Blocked`",

then set *element*'s frozen base URL[p183] to *document*'s fallback base URL[p101] and return.

4. Set *element*'s frozen base URL[p183] to *urlRecord*.
5. Respond to base URL changes[p102] given *document*.

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[p183] 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[p101], and *document*'s character encoding. (Thus, the base[p182] element isn't affected by other base[p182] elements or itself.)
4. If *urlRecord* is failure, return *url*.
5. Return the serialization of *urlRecord*.

Example

In this example, a base[p182] element is used to set the document base URL[p101]:

```
<!DOCTYPE html>
<html lang="en">
    <head>
        <title>This is an example for the &lt;base&gt; element</title>
        <base href="https://www.example.com/news/index.html">
    </head>
    <body>
        <p>Visit the <a href="archives.html">archives</a>.</p>
    </body>
</html>
```

The link in the above example would be a link to "https://www.example.com/news/archives.html".

4.2.4 The link element §[p184]

Categories[p154]:

Metadata content[p156].
If the element is allowed in the body[p186]: flow content[p157].
If the element is allowed in the body[p186]: phrasing content[p157].

Contexts in which this element can be used[p154]:

Where metadata content[p156] is expected.
In a noscript[p703] element that is a child of a head[p180] element.
If the element is allowed in the body[p186]: where phrasing content[p157] is expected.

Content model[p154]:

Nothing[p155].

Tag omission in text/html[p154]:

No end tag[p1359].

Content attributes[p154]:

Global attributes[p162]
href[p185] — Address of the hyperlink[p313]
crossorigin[p186] — How the element handles crossorigin requests
rel[p185] — Relationship between the document containing the hyperlink[p313] and the destination resource
media[p186] — Applicable media
integrity[p186] — Integrity metadata used in *Subresource Integrity* checks [SRI][p1586]
hreflang[p186] — Language of the linked resource
type[p186] — Hint for the type of the referenced resource

referrerpolicy[p187] — Referrer policy for fetches initiated by the element
sizes[p188] — Sizes of the icons (for rel[p185]="icon[p332]")
imagesrcset[p187] — Images to use in different situations, e.g., high-resolution displays, small monitors, etc. (for rel[p185]="preload[p340]")
imagesizes[p187] — Image sizes for different page layouts (for rel[p185]="preload[p340]")
as[p188] — Destination for a preload request (for rel[p185]="preload[p340]" and rel[p185]="modulepreload[p335]")
blocking[p188] — Whether the element is potentially render-blocking[p107]
color[p188] — Color to use when customizing a site's icon (for rel[p185]="mask-icon")
disabled[p188] — Whether the link is disabled
fetchpriority[p189] — Sets the priority for fetches initiated by the element
Also, the title[p187] attribute has special semantics[p187] on this element: Title of the link; CSS style sheet set name

Accessibility considerations[p154]:

For authors.
For implementers.

Sanitization[p154]:

Uncategorized[p1249].

DOM interface[p154]:

IDL

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

  [CEReactions, ReflectURL] attribute USVString href;
  [CEReactions] attribute DOMString? crossOrigin;
  [CEReactions, Reflect] attribute DOMString rel;
  [CEReactions] attribute DOMString as;
  [SameObject, PutForwards=value, Reflect="rel"] readonly attribute DOMTokenList relList;
  [CEReactions, Reflect] attribute DOMString media;
  [CEReactions, Reflect] attribute DOMString integrity;
  [CEReactions, Reflect] attribute DOMString hreflang;
  [CEReactions, Reflect] attribute DOMString type;
  [SameObject, PutForwards=value, Reflect] readonly attribute DOMTokenList sizes;
  [CEReactions, Reflect] attribute USVString imageSrcset;
  [CEReactions, Reflect] attribute DOMString imageSizes;
  [CEReactions] attribute DOMString referrerPolicy;
  [SameObject, PutForwards=value, Reflect] readonly attribute DOMTokenList blocking;
  [CEReactions, Reflect] attribute boolean disabled;
  [CEReactions] attribute DOMString fetchPriority;

  // also has obsolete members
};
HTMLLinkElement includes LinkStyle;
```

The link[p184] 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[p185] attribute is present, then its value must be a valid non-empty URL potentially surrounded by spaces[p100]. One or both of the href[p185] or imagesrcset[p187] attributes must be present.

If both the href[p185] and imagesrcset[p187] 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 a unordered set of unique space-separated tokens[p99]. The allowed keywords and their meanings[p326] are defined in a later section. If the rel[p185] 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.

rel[p185]'s supported tokens are the keywords defined in HTML link types[p326] which are allowed on link[p184] elements, impact the processing model, and are supported by the user agent. The possible supported tokens are alternate[p327], dns-prefetch[p330], expect[p330], icon[p332], manifest[p334], modulepreload[p335], next[p348], pingback[p338], preconnect[p339], prefetch[p340], preload[p340], search[p344], and stylesheet[p344]. rel[p185]'s supported tokens must only include the tokens from this list that the user agent implements the processing model for.

Note

Theoretically a user agent could support the processing model for the canonical keyword — if it were a search engine that executed JavaScript. But in practice that's quite unlikely. So in most cases, canonical ought not be included in rel's supported tokens.

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 is expected.

Note

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.

Note

Each link created for a link element is handled separately. For instance, if there are two link elements with `rel="stylesheet"`*, they each count as a separate external resource, and each is affected by its own attributes independently. Similarly, if a single link element has a rel attribute with the value* `next stylesheet`*, it creates both a hyperlink (for the next keyword) and an external resource link (for the stylesheet keyword), and they are affected by other attributes (such as media or title) differently.*

Example

For example, the following link element creates two hyperlinks (to the same page):

```
<link rel="author license" href="/about">
```

The two links created by this element are one whose semantic is that the target page has information about the current page's author, and one whose semantic is that the target page has information regarding the license under which the current page is provided.

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[p313], the type[p186] attribute is used as a hint to user agents so that they can avoid fetching resources they do not support.

The **referrerpolicy** attribute is a referrer policy attribute[p104]. It is intended for use with external resource links[p313], where it helps set the referrer policy used when fetching and processing the linked resource[p190]. [REFERRERPOLICY][p1585]

The **title** attribute gives the title of the link. With one exception, it is purely advisory. The value is text. The exception is for style sheet links that are in a document tree, for which the title[p187] attribute defines CSS style sheet sets.

Note

The title[p187] attribute on link[p184] elements differs from the global title[p164] attribute of most other elements in that a link without a title does not inherit the title of the parent element: it merely has no title.

The **imagesrcset** attribute may be present, and is a srcset attribute[p375].

The imagesrcset[p187] and href[p185] attributes (if width descriptors[p375] are not used) together contribute the image sources[p378] to the source set[p378].

If the imagesrcset[p187] attribute is present and has any image candidate strings[p375] using a width descriptor[p375], the **imagesizes** attribute must also be present, and is a sizes attribute[p376]. The imagesizes[p187] attribute contributes the source size[p378] to the source set[p378].

The imagesrcset[p187] and imagesizes[p187] attributes must only be specified on link[p184] elements that have both a rel[p185] attribute that specifies the preload[p340] keyword, as well as an as[p188] attribute in the "image" state.

Example

These attributes allow preloading the appropriate resource that is later used by an img[p359] element that has the corresponding values for its srcset[p360] and sizes[p361] attributes:

```
<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[p185] attribute, as it would only be relevant for browsers that do not support imagesrcset[p187], and in those cases it would likely cause the incorrect image to be preloaded.

Example

The imagesrcset[p187] attribute can be combined with the media[p186] attribute to preload the appropriate resource selected from a picture[p355] element's sources, for art direction[p370]:

```
<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[p99] which are ASCII case-insensitive. Each value must be either an ASCII case-insensitive match for the string "any[p332]", or a value that consists of two valid non-negative integers[p81] 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[p184] elements that have a rel[p185] attribute that specifies the icon[p332] keyword or the `apple-touch-icon` keyword.

Note

The `apple-touch-icon` keyword is a registered extension to the predefined set of link types[p348], but user agents are not required to support it in any way.

The **as** attribute specifies either a preload destination[p342] or a module preload destination[p335] for a preload request for the resource given by the href[p185] attribute. It is an enumerated attribute[p80]. Each of the union of preload destinations[p342] and module preload destinations[p335] is a keyword for this attribute, mapping to a state of the same name. The attribute must be specified on link[p184] elements that have a rel[p185] attribute that contains the preload[p340] keyword; in such cases it must have a value which is a preload destination[p342]. It may be specified on link[p184] elements that have a rel[p185] attribute that contains the modulepreload[p335] keyword; in such cases it must have a value which is a module preload destination[p335]. For other link[p184] elements, it must not be specified.

The processing model for how the as[p188] attribute is used is given in an individual link type's fetch and process the linked resource[p190] algorithm.

Note

The attribute does not have a missing value default[p80] or invalid value default[p80], meaning that invalid or missing values for the attribute map to no state. This is accounted for in the processing model. For preload[p340] links, both conditions are an error; for modulepreload[p335] links, a missing value will be treated as "`script`".

The **blocking** attribute is a blocking attribute[p107]. It is used by link types stylesheet[p344] and expect[p330], and it must only be specified on link elements that have a rel[p185] attribute containing those keywords.

The **color** attribute is used with the `mask-icon` link type. The attribute must only be specified on link[p184] elements that have a rel[p185] 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[p188] attribute.

Note

The `mask-icon` keyword is a registered extension to the predefined set of link types[p348], but user agents are not required to support it in any way.

link[p184] elements have an associated **explicitly enabled** boolean. It is initially false.

The **disabled** attribute is a boolean attribute[p79] that is used with the stylesheet[p344] link type. The attribute must only be specified on link[p184] elements that have a rel[p185] attribute that contains the stylesheet[p344] keyword.

Whenever the disabled[p188] attribute is removed, set the link[p184] element's explicitly enabled[p188] attribute to true.

Example

Removing the disabled[p188] attribute dynamically, e.g., using `document.querySelector("link").removeAttribute("disabled")`, will fetch and apply the style sheet:

```
<link disabled rel="alternate stylesheet" href="css/pooh">
```

The **fetchpriority** attribute is a fetch priority attribute[p108] that is intended for use with external resource links[p313], where it is used to set the priority used when fetching and processing the linked resource[p190].

Note

There is no reflecting IDL attribute for the color[p188] attribute, but this might be added later.

The **as** IDL attribute must reflect[p108] the as[p188] content attribute, limited to only known values[p109].

The **crossOrigin** IDL attribute must reflect[p108] the crossorigin[p186] content attribute, limited to only known values[p109].

The **referrerPolicy** IDL attribute must reflect[p108] the referrerpolicy[p187] content attribute, limited to only known values[p109].

The **fetchPriority** IDL attribute must reflect[p108] the fetchpriority[p189] content attribute, limited to only known values[p109].

Note

The relList[p185] attribute can be used for feature detection, by calling its supports() method to check which types of links[p326] are supported.

4.2.4.1 Processing the media[p186] attribute §[p18]

If the link is a hyperlink[p313] then the media[p186] attribute is purely advisory, and describes for which media the document in question was designed.

However, if the link is an external resource link[p313], then the media[p186] attribute is prescriptive. The user agent must apply the external resource when the media[p186] attribute's value matches the environment[p100] and the other relevant conditions apply, and must not apply it otherwise.

The default, if the media[p186] 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[p186] attribute §[p18]

If the type[p186] 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[p313] 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[p190]; if the UA does support the given MIME type for the given link relationship, then the UA should fetch and process the linked resource[p190] at the appropriate time as specified for the external resource link[p313]'s particular type. If the attribute is omitted, and the external resource link[p313] type does not have a default type defined, but the user agent would fetch and process the linked resource[p190] if the type was known and supported, then the user agent should fetch and process the linked resource[p190] under the assumption that it will be supported.

User agents must not consider the type[p186] attribute authoritative — upon fetching the resource, user agents must not use the type[p186] 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[p313] type defines rules for processing the resource's Content-Type metadata[p102], 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[p102], 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[p102] to determine the type of the resource. If there is no type metadata, but the external resource link[p313] type has a default type defined, then the user agent must assume that the resource is of that type.

Note

The `stylesheet`[p344] *link type defines rules for processing the resource's* *Content-Type metadata*[p102].

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:

```
<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`[p1577], 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[p102] metadata, or with a syntactically incorrect type like `Content-Type: "null"`, then the default type for `stylesheet`[p344] links would kick in. Since that default type is `text/css`[p1577], the style sheet *would* nonetheless be applied.

4.2.4.3 Fetching and processing a resource from a `link`[p184] element §[p190]

All external resource links[p313] have a **fetch and process the linked resource** algorithm, which takes a `link`[p184] element *el*. They also have **linked resource fetch setup steps** which take a `link`[p184] element *el* and request *request*. Individual link types may provide their own fetch and process the linked resource[p190] algorithm, but unless explicitly stated, they use the default fetch and process the linked resource[p190] algorithm. Similarly, individual link types may provide their own linked resource fetch setup steps[p190], but unless explicitly stated, these steps just return true.

The **default fetch and process the linked resource**, given a `link`[p184] element *el*, is as follows:

1. Let *options* be the result of creating link options[p192] from *el*.
2. Let *request* be the result of creating a link request[p191] given *options*.
3. If *request* is null, then return.
4. Run the linked resource fetch setup steps[p190], given *el* and *request*. If the result is false, then return.
5. Set *request*'s initiator type to "`css`" if *el*'s `rel`[p185] attribute contains the keyword `stylesheet`[p344]; "`link`" otherwise.
6. Fetch *request* with *processResponseConsumeBody* set to the following steps given response *response* and null, failure, or a byte sequence *bodyBytes*:
    1. Let *success* be true.
    2. If any of the following are true:
        - *bodyBytes* is null or failure; or
        - *response*'s status is not an ok status,

        then set *success* to false.

        Note

        Note that content-specific errors, e.g., CSS parse errors or PNG decoding errors, do not affect success.
    3. Otherwise, wait for the link resource[p313]'s critical subresources[p46] to finish loading.

        The specification that defines a link type's critical subresources[p46] (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[p313]'s critical subresources[p46] to be fetched and processed, with the expectation that this will be done correctly.

4. Process the linked resource[p191] given *el*, *success*, *response*, and *bodyBytes*.

To **create a link request** given a link processing options[p191] *options*:

1. Assert: *options*'s href[p192] is not the empty string.
2. Let *url* be the result of encoding-parsing a URL[p101] given *options*'s href[p192], relative to *options*'s base URL[p192].

   Passing the base URL instead of a document or environment is tracked by issue #9715.
3. If *url* is failure, then return null.
4. Let *request* be the result of creating a potential-CORS request[p102] given *url*, *options*'s destination[p192], and *options*'s crossorigin[p192].
5. Set *request*'s policy container to *options*'s policy container[p192].
6. Set *request*'s integrity metadata to *options*'s integrity[p192].
7. Set *request*'s cryptographic nonce metadata to *options*'s cryptographic nonce metadata[p192].
8. Set *request*'s referrer policy to *options*'s referrer policy[p192].
9. Set *request*'s client to *options*'s environment[p192].
10. Set *request*'s priority to *options*'s fetch priority[p192].
11. Return *request*.

User agents may opt to only try to fetch and process[p190] such resources when they are needed, instead of pro-actively fetching all the external resources[p313] that are not applied.

Similar to the fetch and process the linked resource[p190] algorithm, all external resource links[p313] have a **process the linked resource** algorithm which takes a `link`[p184] element *el*, boolean *success*, a response *response*, and a byte sequence *bodyBytes*. Individual link types may provide their own process the linked resource[p191] algorithm, but unless explicitly stated, that algorithm does nothing.

Unless otherwise specified for a given `rel`[p185] keyword, the element must delay the load event[p1458] of the element's node document until all the attempts to fetch and process the linked resource[p190] and its critical subresources[p46] 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[p1458].)

4.2.4.4 Processing `Link` headers [§](p19)

All link types that can be external resource links[p313] define a **process a link header** algorithm, which takes a link processing options[p191]. This algorithm defines whether and how they react to appearing in an HTTP `Link` response header.

Note

For most link types, this algorithm does nothing. The summary table[p326] is a good reference to quickly know whether a link type has defined process a link header[p191] steps.

A **link processing options** is a struct. It has the following items:

***href* (default the empty string)**

***initiator* (default "`link`")**

***integrity* (default the empty string)**

***type* (default the empty string)**

***cryptographic nonce metadata* (default the empty string)**

A string

***destination* (default the empty string)**

A destination type.

***crossorigin* (default No CORS[p103])**

A CORS settings attribute[p103] state

***referrer policy* (default the empty string)**

A referrer policy

***source set* (default null)**

Null or a source set[p378]

base URL

A URL

origin

An origin[p939]

environment

An environment[p1143]

policy container

A policy container[p960]

***document* (default null)**

Null or a `Document`[p135]

***on document ready* (default null)**

Null or an algorithm accepting a `Document`[p135]

***fetch priority* (default `Auto`[p108])**

A fetch priority attribute[p108] state

Note

A link processing options[p191] has a base URL[p192] and an href[p192] rather than a parsed URL because the URL could be a result of the options's source set[p192].

To **create link options from element** given a `link`[p184] element *el*:

1. Let *document* be *el*'s node document.
2. Let *options* be a new link processing options[p191] with

   crossorigin[p192]
   : the state of *el*'s `crossorigin`[p186] content attribute

   referrer policy[p192]
   : the state of *el*'s `referrerpolicy`[p187] content attribute

   source set[p192]
   : *el*'s source set[p378]

   base URL[p192]
   : *document*'s document base URL[p101]

   origin[p192]
   : *document*'s origin

   environment[p192]
   : *document*'s relevant settings object[p1151]

   policy container[p192]
   : *document*'s policy container[p136]

document[p192]
document
cryptographic nonce metadata[p192]
the current value of *el*'s [[CryptographicNonce]][p104] internal slot
fetch priority[p192]
the state of *el*'s `fetchpriority`[p189] content attribute

3. If *el* has an `href`[p185] attribute, then set *options*'s href[p192] to the value of *el*'s `href`[p185] attribute.
4. If *el* has an `integrity`[p186] attribute, then set *options*'s integrity[p192] to the value of *el*'s `integrity`[p186] content attribute.
5. If *el* has a `type`[p186] attribute, then set *options*'s type[p192] to the value of *el*'s `type`[p186] attribute.
6. Assert: *options*'s href[p192] is not the empty string, or *options*'s source set[p192] is not null.

   A `link`[p184] element with neither an `href`[p185] or an `imagesrcset`[p187] does not represent a link.
7. Return *options*.

To **extract links from headers** given a header list *headers*:

1. Let *links* be a new list.
2. Let *rawLinkHeaders* be the result of getting, decoding, and splitting `` `Link`[p184] `` from *headers*.
3. For each *linkHeader* of *rawLinkHeaders*:
   1. Let *linkObject* be the result of parsing *linkHeader*. [WEBLINK][p1588]
   2. If *linkObject*["`target_uri`"] does not exist, then continue.
   3. Append *linkObject* to *links*.
4. Return *links*.

To **process link headers** given a `Document`[p135] *doc*, a response *response*, and a "`pre-media`" or "`media`" *phase*:

1. Let *links* be the result of extracting links[p193] from *response*'s header list.
2. For each *linkObject* in *links*:
   1. Let *rel* be *linkObject*["`relation_type`"].
   2. Let *attribs* be *linkObject*["`target_attributes`"].
   3. Let *expectedPhase* be "`media`" if either "`srcset`[p360]", "`imagesrcset`[p187]", or "`media`[p186]" exist in *attribs*; otherwise "`pre-media`".
   4. If *expectedPhase* is not *phase*, then continue.
   5. If *attribs*["`media`[p186]"] exists and *attribs*["`media`[p186]"] does not match the environment[p100], then continue.
   6. Let *options* be a new link processing options[p191] with

      href[p192]
      linkObject["`target_uri`"]
      base URL[p192]
      doc's document base URL[p101]
      origin[p192]
      doc's origin
      environment[p192]
      doc's relevant settings object[p1151]
      policy container[p192]
      doc's policy container[p136]
      document[p192]
      doc
   7. Apply link options from parsed header attributes[p194] to *options* given *attribs* and *rel*. If that returned false, then return.

8. If *attribs*["imagesrcset[p187]"] exists and *attribs*["imagesizes[p187]"] exists, then set *options*'s source set[p192] to the result of creating a source set[p385] given *linkObject*["target_uri"], *attribs*["imagesrcset[p187]"], *attribs*["imagesizes[p187]"], and null.

9. Run the process a link header[p191] steps for *rel* given *options*.

To **apply link options from parsed header attributes** to a link processing options[p191] *options* given *attribs* and a string *rel*:

1. If *rel* is "preload[p340]":

    1. If *attribs*["as[p188]"] does not exist, then return false.

    2. Let *destination* be the result of translating[p342] *attribs*["as[p188]"].

    3. If *destination* is null, then return false.

    4. Set *options*'s destination[p192] to *destination*.

2. If *attribs*["crossorigin[p186]"] exists and is an ASCII case-insensitive match for one of the CORS settings attribute[p103] keywords[p80], then set *options*'s crossorigin[p192] to the CORS settings attribute[p103] state corresponding to that keyword.

3. If *attribs*["integrity[p186]"] exists, then set *options*'s integrity[p192] to *attribs*["integrity[p186]"].

4. If *attribs*["referrerpolicy[p187]"] exists and is an ASCII case-insensitive match for some referrer policy, then set *options*'s referrer policy[p192] to that referrer policy.

5. If *attribs*["nonce[p104]"] exists, then set *options*'s nonce[p192] to *attribs*["nonce[p104]"].

6. If *attribs*["type[p186]"] exists, then set *options*'s type[p192] to *attribs*["type[p186]"].

7. If *attribs*["fetchpriority[p189]"] exists and is an ASCII case-insensitive match for a fetch priority attribute[p108] keyword, then set *options*'s fetch priority[p192] to that fetch priority attribute[p108] keyword.

8. Return true.

MDN

4.2.4.5 Early hints §[p194]

Early hints allow user-agents to perform some operations, such as to speculatively load resources that are likely to be used by the document, before the navigation request is fully handled by the server and a response code is served. Servers can indicate early hints by serving a response with a 103 status code before serving the final response.[RFC8297][p1586]

Note

For compatibility reasons early hints are typically delivered over HTTP/2 or above, but for readability we use HTTP/1.1-style notation below.

Example

For example, given the following sequence of responses:

```
103 Early Hint
Link: </image.png>; rel=preload; as=image

200 OK
Content-Type: text/html

<!DOCTYPE html>
...
<img src="/image.png">
```

the image will start loading before the HTML content arrives.

Note

Only the first early hint response served during the navigation is handled, and it is discarded if it is succeeded by a cross-origin redirect.

In addition to the `Link` headers, it is possible that the 103 response contains a Content Security Policy header, which is enforced when processing the early hint.

Example

For example, given the following sequence of responses:

```
103 Early Hint
Content-Security-Policy: style-src: self;
Link: </style.css>; rel=preload; as=style

103 Early Hint
Link: </image.png>; rel=preload; as=image

302 Redirect
Location: /alternate.html

200 OK
Content-Security-Policy: style-src: none;
Link: </font.ttf>; rel=preload; as=font
```

The font and style would be loaded, and the image will be discarded, as only the first early hint response in the final redirect chain is respected. The late Content Security Policy header comes after the request to fetch the style has already been performed, but the style will not be accessible to the document.

To **process early hint headers** given a response *response* and an environment[p1143] *reservedEnvironment*:

Note

Early-hint `Link` headers are always processed before `Link` headers from the final response, followed by `link`[p184] elements. This is equivalent to prepending the contents of the early and final `Link` headers to the `Document`[p135]'s `head`[p180] element, in respective order.

1. Let *earlyPolicyContainer* be the result of creating a policy container from a fetch response[p960] given *response* and *reservedEnvironment*.

   Note

   This allows the early hint response to include a Content Security Policy which would be enforced when fetching the early hint request.

2. Let *links* be the result of extracting links[p193] from *response*'s header list.

3. Let *earlyHints* be an empty list.

4. For each *linkObject* in *links*:

   Note

   The moment we receive the early hint link header, we begin fetching earlyRequest. *If it comes back before the `Document`[p135] is created, we set* earlyResponse *to the response of that fetch and once the `Document`[p135] is created we commit it (by making it available in the map of preloaded resources[p341] as if it was a `link`[p184] element). If the `Document`[p135] is created first, the response is committed as soon as it becomes available.*

   1. Let *rel* be *linkObject*["`relation_type`"].

   2. Let *options* be a new link processing options[p191] with

      href[p192]
      : *linkObject*["`target_uri`"]

      initiator[p192]
      : "`early-hint`"

      base URL[p192]
      : *response*'s URL

      origin[p192]
      : *response*'s URL's origin

      environment[p192]
      : *reservedEnvironment*

policy container[p192]
earlyPolicyContainer

3. Let *attribs* be *linkObject*["target_attributes"].

> **Note**
> *Only the* as[p188], crossorigin[p186], integrity[p186], *and* type[p186] *attributes are handled as part of early hint processing. The other ones, in particular* blocking[p188], imagesrcset[p187], imagesizes[p187], *and* media[p186] *are only applicable once a* Document[p135] *is created.*

4. Apply link options from parsed header attributes[p194] to *options* given *attribs* and *rel*. If that returned false, then return.

5. Run the process a link header[p191] steps for *rel* given *options*.

6. Append *options* to *earlyHints*.

5. Return the following substeps given Document[p135] *doc*: for each *options* in *earlyHints*:

    1. If *options*'s on document ready[p192] is null, then set *options*'s document[p192] to *doc*.

    2. Otherwise, call *options*'s on document ready[p192] with *doc*.

4.2.4.6 Providing users with a means to follow hyperlinks created using the link[p184] element §[p19]

Interactive user agents may provide users with a means to follow the hyperlinks[p321] created using the link[p184] element, somewhere within their user interface. Such invocations of the follow the hyperlink[p321] algorithm must set the *userInvolvement*[p321] argument to "browser UI[p1062]". 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[p313] created with each link[p184] element in the document:

- The relationship between this document and the resource (given by the rel[p185] attribute)
- The title of the resource (given by the title[p187] attribute).
- The address of the resource (given by the href[p185] attribute).
- The language of the resource (given by the hreflang[p186] attribute).
- The optimum media for the resource (given by the media[p186] attribute).

User agents could also include other information, such as the type of the resource (as given by the type[p186] attribute).

4.2.5 The meta element §[p19]

Categories[p154]**:**
Metadata content[p156].
If the itemprop[p833] attribute is present: flow content[p157].
If the itemprop[p833] attribute is present: phrasing content[p157].

Contexts in which this element can be used[p154]**:**
If the charset[p197] attribute is present, or if the element's http-equiv[p202] attribute is in the Encoding declaration state[p203]: in a head[p180] element.
If the http-equiv[p202] attribute is present but not in the Encoding declaration state[p203]: in a head[p180] element.
If the http-equiv[p202] attribute is present but not in the Encoding declaration state[p203]: in a noscript[p703] element that is a child of a head[p180] element.
If the name[p197] attribute is present: where metadata content[p156] is expected.
If the itemprop[p833] attribute is present: where metadata content[p156] is expected.
If the itemprop[p833] attribute is present: where phrasing content[p157] is expected.

Content model[p154]**:**
Nothing[p155].

Tag omission in text/html[p154]**:**

No end tag[p1359].

Content attributes[p154]**:**

Global attributes[p162]
name[p197] — Metadata name
http-equiv[p202] — Pragma directive
content[p197] — Value of the element
charset[p197] — Character encoding declaration[p206]
media[p197] — Applicable media

Accessibility considerations[p154]**:**

For authors.
For implementers.

Sanitization[p154]**:**

Uncategorized[p1249].

DOM interface[p154]**:**

IDL

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

  [CEReactions, Reflect] attribute DOMString name;
  [CEReactions, Reflect="http-equiv"] attribute DOMString httpEquiv;
  [CEReactions, Reflect] attribute DOMString content;
  [CEReactions, Reflect] attribute DOMString media;

  // also has obsolete members
};
```

The meta[p196] element represents[p148] various kinds of metadata that cannot be expressed using the title[p181], base[p182], link[p184], style[p207], and script[p685] elements.

The meta[p196] element can represent document-level metadata with the name[p197] attribute, pragma directives with the http-equiv[p202] attribute, and the file's character encoding declaration[p206] when an HTML document is serialized to string form (e.g. for transmission over the network or for disk storage) with the charset[p197] attribute.

Exactly one of the name[p197], http-equiv[p202], charset[p197], and itemprop[p833] attributes must be specified.

If either name[p197], http-equiv[p202], or itemprop[p833] is specified, then the content[p197] 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[p206]. If the attribute is present, its value must be an ASCII case-insensitive match for the string "utf-8".

Note

The charset[p197] attribute on the meta[p196] element has no effect in XML documents, but is allowed in XML documents in order to facilitate migration to and from XML.

There must not be more than one meta[p196] element with a charset[p197] 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[p196] element has a **name** attribute, it sets document metadata. Document metadata is expressed in terms of name-value pairs, the name[p197] attribute on the meta[p196] element giving the name, and the content[p197] 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[p196] element has no content[p197] attribute, then the value part of the metadata name-value pair is the empty string.

The **media** attribute says which media the metadata applies to. The value must be a valid media query list[p100]. Unless the name[p197] is theme-color[p200], the media[p197] attribute has no effect on the processing model and must not be used by authors.

4.2.5.1 Standard metadata names §p198

This specification defines a few names for the name[p197] attribute of the meta[p196] 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[p198] metadata name must not be used. Translations of the web application's name may be given, using the lang[p165] attribute to specify the language of each name.

There must not be more than one meta[p196] element with a given language[p166] and where the name[p197] attribute value is an ASCII case-insensitive match for application-name[p198] per document.

User agents may use the application name in UI in preference to the page's title[p181], 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[p166] of the Document[p135]'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[p196] element in the Document[p135] where the name[p197] attribute value is an ASCII case-insensitive match for application-name[p198] and whose language[p166] is the language in question.

   If none of the languages have such a meta[p196] element, then return; there's no given application name.
5. Return the value of the content[p197] attribute of the first meta[p196] element in the Document[p135] in tree order where the name[p197] attribute value is an ASCII case-insensitive match for application-name[p198] and whose language[p166] 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[p196] element where the name[p197] attribute value is an ASCII case-insensitive match for description[p198] 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[p180] element, to identify itself as the tool used to generate the page:

```
<meta name=generator content="Frontweaver 8.2">
```

keywords

The value must be a set of comma-separated tokens[p99], each of which is a keyword relevant to the page.

Example

This page about typefaces on British motorways uses a meta[p196] element to specify some keywords that users might use to look for the page:

```
<!DOCTYPE HTML>
<html lang="en-GB">
 <head>
  <title>Typefaces on UK motorways</title>
  <meta name="keywords" content="british,type face,font,fonts,highway,highways">
 </head>
 <body>
  ...
```

Note

Many search engines do not consider such keywords, because this feature has historically been used unreliably and even misleadingly as a way to spam search engine results in a way that is not helpful for users.

To obtain the list of keywords that the author has specified as applicable to the page, the user agent must run the following steps:

1. Let *keywords* be an empty list.
2. For each meta[p196] element with a name[p197] attribute and a content[p197] attribute and where the name[p197] attribute value is an ASCII case-insensitive match for keywords[p198]:
    1. Split the value of the element's content attribute on commas.
    2. Add the resulting tokens, if any, to *keywords*.
3. Remove any duplicates from *keywords*.
4. Return *keywords*. This is the list of keywords that the author has specified as applicable to the page.

User agents should not use this information when there is insufficient confidence in the reliability of the value.

Example

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 referrer policy, which defines the default referrer policy for the Document[p135]. [REFERRERPOLICY][p1585]

If any meta[p196] element *element* is inserted into the document[p47], or has its name[p197] or content[p197] attributes changed, user agents must run the following algorithm:

1. If *element* is not in a document tree, then return.
2. If *element* does not have a name[p197] attribute whose value is an ASCII case-insensitive match for "referrer[p199]", then return.
3. If *element* does not have a content[p197] attribute, or that attribute's value is the empty string, then return.
4. Let *value* be the value of *element*'s content[p197] attribute, converted to ASCII lowercase.
5. If *value* is one of the values given in the first column of the following table, then set *value* to the value given in the second column:

Legacy value	Referrer policy
never	no-referrer
default	the default referrer policy
always	unsafe-url
origin-when-crossorigin	origin-when-cross-origin

6. If *value* is a referrer policy, then set *element*'s node document's policy container[p136]'s referrer policy[p960] to *policy*.

Note

For historical reasons, unlike other standard metadata names, the processing model for `referrer`[p199] is not responsive to element removals, and does not use tree order. Only the most-recently-inserted or most-recently-modified `meta`[p196] element in this state has an effect.

MDN

`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.

Within an HTML document, the `media`[p197] attribute value must be unique amongst all the `meta`[p196] elements with their `name`[p197] attribute value set to an ASCII case-insensitive match for `theme-color`[p200].

Example

This standard itself uses "WHATWG green" as its theme color:

```
<!DOCTYPE HTML>
<title>HTML Standard</title>
<meta name="theme-color" content="#3c790a">
...
```

The `media`[p197] attribute may be used to describe the context in which the provided color should be used.

Example

If we only wanted to use "WHATWG green" as this standard's theme color in dark mode, we could use the `prefers-color-scheme` media feature:

```
<!DOCTYPE HTML>
<title>HTML Standard</title>
<meta name="theme-color" content="#3c790a" media="(prefers-color-scheme: dark)">
...
```

To obtain a page's theme color, user agents must run the following steps:

1. Let *candidate elements* be the list of all `meta`[p196] elements that meet the following criteria, in tree order:
   - the element is in a document tree;
   - the element has a `name`[p197] attribute, whose value is an ASCII case-insensitive match for `theme-color`[p200]; and
   - the element has a `content`[p197] attribute.
2. For each *element* in *candidate elements*:
   1. If *element* has a `media`[p186] attribute and the value of *element*'s `media`[p197] attribute does not match the environment[p100], then continue.
   2. Let *value* be the result of stripping leading and trailing ASCII whitespace from the value of *element*'s `content`[p197] attribute.
   3. Let *color* be the result of parsing *value*.
   4. If *color* is not failure, then return *color*.
3. Return nothing (the page has no theme color).

If any `meta`[p196] elements are inserted into the document[p47] or removed from the document[p47], or existing `meta`[p196] elements have their `name`[p197], `content`[p197], or `media`[p186] attributes changed, or if the environment changes such that any `meta`[p196] element's `media`[p186] attribute's value may now or may no longer match the environment[p100], 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`[p196] element.

The value must be a string that matches the syntax for the CSS 'color-scheme' property value. It determines the page's supported color-schemes.

There must not be more than one `meta`[p196] element with its `name`[p197] attribute value set to an ASCII case-insensitive match for `color-scheme`[p201] 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:

```
<meta name="color-scheme" content="dark">
```

To obtain a page's supported color-schemes, user agents must run the following steps:

1. Let *candidate elements* be the list of all `meta`[p196] elements that meet the following criteria, in tree order:
    - the element is in a document tree;
    - the element has a `name`[p197] attribute, whose value is an ASCII case-insensitive match for `color-scheme`[p201]; and
    - the element has a `content`[p197] 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 `content`[p197] attribute.
    2. If *parsed* is a valid CSS 'color-scheme' property value, then return *parsed*.
3. Return null.

If any `meta`[p196] elements are inserted into the document[p47] or removed from the document[p47], or existing `meta`[p196] elements have their `name`[p197] or `content`[p197] attributes changed, user agents must re-run the above algorithm.

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 §[p201]

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 `content`[p197] attribute is a URL; in those cases, registering it as an extension to the predefined set of link types[p348] 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 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. [WHATWGWIKI][p1588]

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 §[p20]

When the **http-equiv** attribute is specified on a meta[p196] element, the element is a pragma directive.

⚠Warning!

Despite the name http-equiv[p202], pragma directives are almost entirely unrelated to HTTP headers. Implementers and web developers are best off thinking of them as entirely separate, and the name as being a historical accident.

In more detail, although the refresh[p204] keyword has the same processing model as the corresponding `Refresh[p1137]` header, every other standardized pragma directive has at least slightly different behavior than the similarly-named header. (And usually dramatically-different behavior.)

Implementers or specification writers contemplating adding new document-level pragmas or HTTP header-controlled switches should be cautious about this mismatch, and avoid perpetuating the existing confusion by adding the same or similar behavior to both an HTTP header and an http-equiv[p202] pragma. Instead, consider providing only an HTTP header, or if an in-document pragma is needed, consider adding a new attribute to meta[p196] similar to the model used by the charset[p197] attribute. (Note that avoiding in-document pragmas is often the better

choice, since the DOM is mutable. Thus, even in the simple case where the developer does not add, remove, or mutate `meta`[p196] elements, the policy will go from un-applied to applied during parsing, which can have complex implications.)

The `http-equiv`[p202] attribute is an enumerated attribute[p80] with the following keywords and states:

Keyword	Conforming	State	Brief description
`content-language`	No	Content language[p203]	Sets the pragma-set default language[p203].
`content-type`		Encoding declaration[p203]	An alternative form of setting the `charset`[p197].
`default-style`		Default style[p204]	Sets the name of the default CSS style sheet set.
`refresh`		Refresh[p204]	Acts as a timed redirect.
`set-cookie`	No	Set-Cookie[p206]	Has no effect.
`x-ua-compatible`		X-UA-Compatible[p206]	In practice, encourages Internet Explorer to more closely follow the specifications.
`content-security-policy`		Content security policy[p206]	Enforces a Content Security Policy on a `Document`[p135].

When a `meta`[p196] element is inserted into the document[p47], if its `http-equiv`[p202] 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`[p203]`"`)**

Note

This feature is non-conforming. Authors are encouraged to use the `lang`[p165] attribute instead.

This pragma sets the **pragma-set default language**. Until such a pragma is successfully processed, there is no pragma-set default language[p203].

1. If the `meta`[p196] element has no `content`[p197] attribute, then return.
2. If the element's `content`[p197] attribute contains a U+002C COMMA character (,), then return.
3. Let *input* be the value of the element's `content`[p197] 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[p203] to *candidate*.

Note

If the value consists of multiple space-separated tokens, tokens after the first are ignored.

Note

This pragma is almost, but not quite, entirely unlike the HTTP `Content-Language` header of the same name. [HTTP][p1583]

***Encoding declaration state* (`http-equiv="content-type`[p203]`"`)**

The Encoding declaration state[p203] is just an alternative form of setting the `charset`[p197] attribute: it is a character encoding declaration[p206]. This state's user agent requirements are all handled by the parsing section of the specification.

For `meta`[p196] elements with an `http-equiv`[p202] attribute in the Encoding declaration state[p203], the `content`[p197] attribute must have a value that is an ASCII case-insensitive match for a string that consists of: "`text/html;`", optionally followed by any number of ASCII whitespace, followed by "`charset=utf-8`".

A document must not contain both a `meta`[p196] element with an `http-equiv`[p202] attribute in the Encoding declaration state[p203] and a `meta`[p196] element with the `charset`[p197] attribute present.

The Encoding declaration state[p203] may be used in HTML documents, but elements with an http-equiv[p202] attribute in that state must not be used in XML documents.

Default style state **(`http-equiv="default-style[p203]"`)**

This pragma sets the name of the default CSS style sheet set.

1. If the `meta`[p196] element has no `content`[p197] 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`[p197] attribute. [CSSOM][p1581]

Refresh state **(`http-equiv="refresh[p203]"`)**

This pragma acts as a timed redirect.

A `Document`[p135] object has an associated **will declaratively refresh** (a boolean). It is initially false.

1. If the `meta`[p196] element has no `content`[p197] attribute, or if that attribute's value is the empty string, then return.
2. Let *input* be the value of the element's `content`[p197] attribute.
3. Run the shared declarative refresh steps[p204] with the `meta`[p196] element's node document, *input*, and the `meta`[p196] element.

The **shared declarative refresh steps**, given a `Document`[p135] object *document*, string *input*, and optionally a `meta`[p196] element *meta*, are as follows:

1. If *document*'s will declaratively refresh[p204] 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 *timeString* be the result.
6. If *timeString* is the empty string:
    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[p81].
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*:
    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*:
    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. Otherwise, jump to the step labeled *parse*.

7. Skip ASCII whitespace within *input* given *position*.

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*: Set *urlRecord* to the result of encoding-parsing a URL[p101] given *urlString*, relative to *document*.

12. If *urlRecord* is failure, then return.

13. If *urlRecord*'s scheme is "`javascript`[p1068]", then return.

12. Set *document*'s will declaratively refresh[p204] 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[p959] does not have the sandboxed automatic features browsing context flag[p958] set, then navigate[p1063] *document*'s node navigable[p1036] to *urlRecord* using *document*, with *historyHandling*[p1063] set to "`replace`[p1062]".

      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[p1114], 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[p47] *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`[p1137] header).*

    - Provide the user with an interface that, when selected, navigates[p1063] *document*'s node navigable[p1036] to *urlRecord* using *document*.

    - 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`[p196] elements with an `http-equiv`[p202] attribute in the Refresh state[p204], the `content`[p197] attribute must have a value consisting either of:

- just a valid non-negative integer[p81], or

- a valid non-negative integer[p81], 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`[p180] element, to ensure that the page automatically reloads from the server every five minutes:

```
<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:

```
<meta http-equiv="Refresh" content="20; URL=page4.html">
```

Set-Cookie state **(`http-equiv="set-cookie"`)**

This pragma is non-conforming and has no effect.

User agents are required to ignore this pragma.

X-UA-Compatible state **(`http-equiv="x-ua-compatible"`)**

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 **(`http-equiv="content-security-policy"`)**

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]

Note

At the time of inserting the `meta` element to the document, it is possible that some resources have already been fetched. For example, images might be stored in the list of available images prior to dynamically inserting a `meta` element with an `http-equiv` attribute in the Content security policy state. Resources that have already been fetched are not guaranteed to be blocked by a Content Security Policy that's enforced late.

Example

A page might choose to mitigate the risk of cross-site scripting attacks by preventing the execution of inline JavaScript, as well as blocking all plugin content, using a policy such as the following:

```
<meta http-equiv="Content-Security-Policy" content="script-src 'self'; object-src 'none'">
```

There must not be more than one `meta` element with any particular state in the document at a time.

4.2.5.4 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[p206], if it exists, specifies an encoding label using an ASCII case-insensitive match for "utf-8". Regardless of whether a character encoding declaration[p206] is present or not, the actual character encoding used to encode the document must be UTF-8. [ENCODING][p1582]

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[p1366] 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[p196] elements, there can only be one meta[p196]-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[p102], and the document is not an iframe srcdoc document[p405], then the encoding must be specified using a meta[p196] element with a charset[p197] attribute or a meta[p196] element with an http-equiv[p202] attribute in the Encoding declaration state[p203].

Note

A character encoding declaration is required (either in the Content-Type metadata[p102] 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[p405], the document must not have a character encoding declaration[p206]. (In this case, the source is already decoded, since it is part of the document that contained the iframe[p404].)

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[p180] element):

```
<meta charset="utf-8">
```

In XML, the XML declaration would be used instead, at the very top of the markup:

```
<?xml version="1.0" encoding="utf-8"?>
```

4.2.6 The style element §[p207]

✓ MDN

Categories[p154]:

Metadata content[p156].

Contexts in which this element can be used[p154]:

Where metadata content[p156] is expected.
In a noscript[p703] element that is a child of a head[p180] element.

Content model[p154]:

Text[p158] that gives a conformant style sheet.

Tag omission in text/html[p154]:

Neither tag is omissible.

Content attributes[p154]:

Global attributes[p162]
media[p208] — Applicable media
blocking[p209] — Whether the element is potentially render-blocking[p107]
Also, the title[p209] attribute has special semantics[p209] on this element: CSS style sheet set name

Accessibility considerations[p154]:

For authors.
For implementers.

Sanitization[p154]:

Uncategorized[p1249].

DOM interface[p154]:

IDL

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

  attribute boolean disabled;
  [CEReactions, Reflect] attribute DOMString media;
  [SameObject, PutForwards=value, Reflect] readonly attribute DOMTokenList blocking;

  // also has obsolete members
};
HTMLStyleElement includes LinkStyle;
```

The style[p207] element allows authors to embed CSS style sheets in their documents. The style[p207] element is one of several inputs to the styling processing model. The element does not represent[p148] content for the user.

The **disabled** getter steps are:

1. If this does not have an associated CSS style sheet, return false.
2. If this's associated CSS style sheet's disabled flag is set, return true.
3. Return false.

The disabled[p208] setter steps are:

1. If this does not have an associated CSS style sheet, return.
2. If the given value is true, set this's associated CSS style sheet's disabled flag. Otherwise, unset this's associated CSS style sheet's disabled flag.

Example

Importantly, disabled[p208] attribute assignments only take effect when the style[p207] element has an associated CSS style sheet:

```
const style = document.createElement('style');
style.disabled = true;
style.textContent = 'body { background-color: red; }';
document.body.append(style);
console.log(style.disabled); // false
```

The **media** attribute says which media the styles apply to. The value must be a valid media query list[p100]. The user agent must apply the styles when the media[p208] attribute's value matches the environment[p100] and the other relevant conditions apply, and must not apply them otherwise.

Note

The styles might be further limited in scope, e.g. in CSS with the use of `@media` blocks. This specification does not override such further restrictions or requirements.

The default, if the media attribute is omitted, is "all", meaning that by default styles apply to all media.

The **blocking** attribute is a blocking attribute.

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 title attribute on style elements, like the title attribute on link elements, differs from the global title attribute in that a style block without a title does not inherit the title of the parent element: it merely has no title.

The child text content of a style element must be that of a conformant style sheet.

A style element is implicitly potentially render-blocking if the element was created by its node document's parser.

The user agent must run the update a style block algorithm whenever any 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 is not on the stack of open elements of an HTML parser or XML parser, and its 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 *element*, "style", and *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.

This doesn't seem right. Presumably we should be using the element's child text content? Tracked as issue #2997.

7. If *element* contributes a script-blocking style sheet[p211], append *element* to its node document's script-blocking style sheet set[p212].

8. If *element*'s `media`[p208] attribute's value matches the environment[p100] and *element* is potentially render-blocking[p107], then block rendering[p142] on *element*.

Once the attempts to obtain the style sheet's critical subresources[p46], if any, are complete, or, if the style sheet has no critical subresources[p46], once the style sheet has been parsed and processed, the user agent must run these steps:

Fetching the critical subresources[p46] is not well-defined; probably issue #968 is the best resolution for that. In the meantime, any critical subresource[p46] request should have its render-blocking set to whether or not the `style`[p207] element is currently render-blocking[p142].

1. Let *element* be the `style`[p207] element associated with the style sheet in question.

2. Let *success* be true.

3. If the attempts to obtain any of the style sheet's critical subresources[p46] failed for any reason (e.g., DNS error, HTTP 404 response, a connection being prematurely closed, unsupported Content-Type), set *success* to false.

Note

Note that content-specific errors, e.g., CSS parse errors or PNG decoding errors, do not affect success.

4. Queue an element task[p1195] on the networking task source[p1204] given *element* and the following steps:

    1. If *success* is true, fire an event named `load`[p1575] at *element*.

    2. Otherwise, fire an event named `error`[p1575] at *element*.

    3. If *element* contributes a script-blocking style sheet[p211]:

        1. Assert: *element*'s node document's script-blocking style sheet set[p212] contains *element*.

        2. Remove *element* from its node document's script-blocking style sheet set[p212].

    4. Unblock rendering[p142] on *element*.

The element must delay the load event[p1458] of the element's node document until all the attempts to obtain the style sheet's critical subresources[p46], if any, are complete.

Note

This specification does not specify a style system, but CSS is expected to be supported by most web browsers. [CSS][p1580]

The `LinkStyle` interface is also implemented by this element. [CSSOM][p1581]

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.

```
<!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 <em>favorite</em> book of all time has <em>got</em> to be
  <cite>A Cat's Life</cite>. It is a book by P. Rahmel that talks
  about the <i lang="la">Felis catus</i> in modern human society.</p>
 </body>
</html>
```

4.2.7 Interactions of styling and scripting §[p21]

If the style sheet referenced no other resources (e.g., it was an internal style sheet given by a `style`[p207] element with no `@import` rules), then the style rules must be immediately[p44] made available to script; otherwise, the style rules must only be made available to script once the event loop[p1193] reaches its update the rendering[p1198] step.

An element *el* in the context of a `Document`[p135] of an HTML parser[p1368] or XML parser[p1484] **contributes a script-blocking style sheet** if all of the following are true:

- *el* was created by that `Document`[p135]'s parser.
- *el* is either a `style`[p207] element or a `link`[p184] element that was an external resource link that contributes to the styling processing model[p344] when the *el* was created by the parser.
- *el*'s `media` attribute's value matches the environment[p100].
- *el*'s style sheet was enabled when the element was created by the parser.
- The last time the event loop[p1193] reached step 1[p1196], *el*'s root was that `Document`[p135].
- 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. However, this has not yet been thoroughly investigated.

A Document[p135] has a **script-blocking style sheet set**, which is an ordered set, initially empty.

A Document[p135] *document* **has a style sheet that is blocking scripts** if the following steps return true:

1. If *document*'s script-blocking style sheet set[p212] is not empty, then return true.
2. If *document*'s node navigable[p1036] is null, then return false.
3. Let *containerDocument* be *document*'s node navigable[p1036]'s container document[p1038].
4. If *containerDocument* is non-null and *containerDocument*'s script-blocking style sheet set[p212] is not empty, then return true.
5. Return false.

A Document[p135] **has no style sheet that is blocking scripts** if it does not have a style sheet that is blocking scripts[p212].

4.3 Sections §[p212]

4.3.1 The body element §[p212]

Categories[p154]:

None.

Contexts in which this element can be used[p154]:

As the second element in an html[p179] element.

Content model[p154]:

Flow content[p157].

Tag omission in text/html[p154]:

A body[p212] element's start tag[p1358] can be omitted if the element is empty, or if the first thing inside the body[p212] element is not ASCII whitespace or a comment[p1367], except if the first thing inside the body[p212] element is a meta[p196], noscript[p703], link[p184], script[p685], style[p207], or template[p705] element.
A body[p212] element's end tag[p1359] can be omitted if the body[p212] element is not immediately followed by a comment[p1367].

Content attributes[p154]:

Global attributes[p162]
onafterprint[p1215]
onbeforeprint[p1215]
onbeforeunload[p1215]
onhashchange[p1215]
onlanguagechange[p1215]
onmessage[p1215]
onmessageerror[p1215]
onoffline[p1215]
ononline[p1215]
onpageswap[p1215]
onpagehide[p1215]
onpagereveal[p1215]
onpageshow[p1215]
onpopstate[p1215]
onrejectionhandled[p1215]
onstorage[p1215]
onunhandledrejection[p1215]
onunload[p1215]

Accessibility considerations[p154]:

For authors.
For implementers.

Sanitization[p154]**:**

Default[p1249].

DOM interface[p154]**:**

IDL

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

  // also has obsolete members
};

HTMLBodyElement includes WindowEventHandlers;
```

The body[p212] element represents[p148] the contents of the document.

In conforming documents, there is only one body[p212] element. The document.body[p144] IDL attribute provides scripts with easy access to a document's body[p212] element.

Note

Some DOM operations (for example, parts of the drag and drop[p909] model) are defined in terms of "the body element[p144]". This refers to a particular element in the DOM, as per the definition of the term, and not any arbitrary body[p212] element.

The body[p212] element exposes as event handler content attributes[p1207] a number of the event handlers[p1206] of the Window[p965] object. It also mirrors their event handler IDL attributes[p1207].

The event handlers[p1206] of the Window[p965] object named by the Window-reflecting body element event handler set[p1215], exposed on the body[p212] element, replace the generic event handlers[p1206] with the same names normally supported by HTML elements[p46].

Example

Thus, for example, a bubbling error[p1575] event dispatched on a child of the body element[p144] of a Document[p135] would first trigger the onerror[p1215] event handler content attributes[p1207] of that element, then that of the root html[p179] element, and only *then* would it trigger the onerror[p1215] event handler content attribute[p1207] on the body[p212] element. This is because the event would bubble from the target, to the body[p212], to the html[p179], to the Document[p135], to the Window[p965], and the event handler[p1206] on the body[p212] is watching the Window[p965] not the body[p212]. A regular event listener attached to the body[p212] using `addEventListener()`, however, would be run when the event bubbled through the body[p212] and not when it reaches the Window[p965] object.

Example

This page updates an indicator to show whether or not the user is online:

```
<!DOCTYPE HTML>
<html lang="en">
 <head>
  <title>Online or offline?</title>
  <script>
   function update(online) {
     document.getElementById('status').textContent =
       online ? 'Online' : 'Offline';
   }
  </script>
 </head>
 <body ononline="update(true)"
       onoffline="update(false)"
       onload="update(navigator.onLine)">
  <p>You are: <span id="status">(Unknown)</span></p>
 </body>
</html>
```

4.3.2 The article element §[p214]

Categories[p154]:

Flow content[p157].
Sectioning content[p157].
Palpable content[p158].

Contexts in which this element can be used[p154]:

Where sectioning content[p157] is expected.

Content model[p154]:

Flow content[p157].

Tag omission in text/html[p154]:

Neither tag is omissible.

Content attributes[p154]:

Global attributes[p162]

Accessibility considerations[p154]:

For authors.
For implementers.

Sanitization[p154]:

Default[p1249].

DOM interface[p154]:

Uses HTMLElement[p149].

The article[p214] element represents[p148] 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[p214] elements are nested, the inner article[p214] 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[p214] elements nested within the article[p214] element for the blog entry.

Author information associated with an article[p214] element (q.v. the address[p230] element) does not apply to nested article[p214] elements.

Note

When used specifically with content to be redistributed in syndication, the article[p214] element is similar in purpose to the `entry` *element in Atom. [ATOM][p1579]*

Note

The schema.org microdata vocabulary can be used to provide the publication date for an article[p214] element, using one of the CreativeWork subtypes.

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[p214], 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).

¶[p214] Example

This example shows a blog post using the article[p214] element, with some schema.org annotations:

```
<article itemscope itemtype="http://schema.org/BlogPosting">
 <header>
  <h2 itemprop="headline">The Very First Rule of Life</h2>
  <p><time itemprop="datePublished" datetime="2009-10-09">3 days ago</time></p>
  <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>
 <p>...</p>
 <footer>
  <a itemprop="discussionUrl" href="?comments=1">Show comments...</a>
 </footer>
</article>
```

Here is that same blog post, but showing some of the comments:

```
<article itemscope itemtype="http://schema.org/BlogPosting">
 <header>
  <h2 itemprop="headline">The Very First Rule of Life</h2>
  <p><time itemprop="datePublished" datetime="2009-10-09">3 days ago</time></p>
  <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>
 <p>...</p>
 <section>
  <h1>Comments</h1>
  <article itemprop="comment" itemscope itemtype="http://schema.org/Comment" id="c1">
   <link itemprop="url" href="#c1">
   <footer>
    <p>Posted by: <span itemprop="creator" itemscope itemtype="http://schema.org/Person">
     <span itemprop="name">George Washington</span>
    </span></p>
    <p><time itemprop="dateCreated" datetime="2009-10-10">15 minutes ago</time></p>
   </footer>
   <p>Yeah! Especially when talking about your lobbyist friends!</p>
  </article>
  <article itemprop="comment" itemscope itemtype="http://schema.org/Comment" id="c2">
   <link itemprop="url" href="#c2">
   <footer>
    <p>Posted by: <span itemprop="creator" itemscope itemtype="http://schema.org/Person">
     <span itemprop="name">George Hammond</span>
    </span></p>
    <p><time itemprop="dateCreated" datetime="2009-10-10">5 minutes ago</time></p>
   </footer>
   <p>Hey, you have the same first name as me.</p>
  </article>
 </section>
</article>
```

Notice the use of footer[p227] to give the information for each comment (such as who wrote it and when): the footer[p227] element *can* appear at the start of its section when appropriate, such as in this case. (Using header[p226] in this case wouldn't be wrong either; it's mostly a matter of authoring preference.)

Example

In this example, article[p214] elements are used to host widgets on a portal page. The widgets are implemented as customized built-in elements[p796] in order to get specific styling and scripted behavior.

```
<!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">
 <h2>Stocks</h2>
 <table>
```

```
  <thead> <tr> <th> Stock <th> Value <th> Delta
  <tbody> <template> <tr> <td> <td> <td> </template>
 </table>
 <p> <input type=button value="Refresh" onclick="this.parentElement.refresh()">
</article>
<article is="news-widget">
 <h2>News</h2>
 <ul>
  <template>
   <li>
    <p><img> <strong></strong>
    <p>
  </template>
 </ul>
 <p> <input type=button value="Refresh" onclick="this.parentElement.refresh()">
</article>
```

4.3.3 The `section` element §[p216]

Categories[p154]:

Flow content[p157].
Sectioning content[p157].
Palpable content[p158].

Contexts in which this element can be used[p154]:

Where sectioning content[p157] is expected.

Content model[p154]:

Flow content[p157].

Tag omission in text/html[p154]:

Neither tag is omissible.

Content attributes[p154]:

Global attributes[p162]

Accessibility considerations[p154]:

For authors.
For implementers.

Sanitization[p154]:

Default[p1249].

DOM interface[p154]:

Uses `HTMLElement`[p149].

The `section`[p216] element represents[p148] 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`[p214] element instead of the `section`[p216] element when it would make sense to syndicate the contents of the element.

¶ p217

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.

```
<article>
 <hgroup>
  <h2>Apples</h2>
  <p>Tasty, delicious fruit!</p>
 </hgroup>
 <p>The apple is the pomaceous fruit of the apple tree.</p>
 <section>
  <h3>Red Delicious</h3>
  <p>These bright red apples are the most common found in many
  supermarkets.</p>
 </section>
 <section>
  <h3>Granny Smith</h3>
  <p>These juicy, green apples make a great filling for
  apple pies.</p>
 </section>
</article>
```

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.)

```
<!DOCTYPE Html>
<Html Lang=En
 ><Head
   ><Title
     >Graduation Ceremony Summer 2022</Title
   ></Head
 ><Body
   ><H1
     >Graduation</H1
   ><Section
     ><H2
       >Ceremony</H2
     ><P
       >Opening Procession</P
     ><P
       >Speech by Valedictorian</P
     ><P
       >Speech by Class President</P
     ><P
       >Presentation of Diplomas</P
     ><P
       >Closing Speech by Headmaster</P
   ></Section
   ><Section
     ><H2
       >Graduates</H2
     ><Ul
       ><Li
         >Molly Carpenter</Li
```

```
      ><Li
        >Anastasia Luccio</Li
      ><Li
        >Ebenezar McCoy</Li
      ><Li
        >Karrin Murphy</Li
      ><Li
        >Thomas Raith</Li
      ><Li
        >Susan Rodriguez</Li
    ></Ul
  ></Section
 ></Body
></Html>
```

Example

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.

```
<style>
 section { border: double medium; margin: 2em; }
 section.chapter h2 { font: 2em Roboto, Helvetica Neue, sans-serif; }
 section.appendix h2 { font: small-caps 2em Roboto, Helvetica Neue, sans-serif; }
</style>
<header>
 <hgroup>
  <h1>My Book</h1>
  <p>A sample with not much content</p>
 </hgroup>
 <p><small>Published by Dummy Publicorp Ltd.</small></p>
</header>
<section class="chapter">
 <h2>My First Chapter</h2>
 <p>This is the first of my chapters. It doesn't say much.</p>
 <p>But it has two paragraphs!</p>
</section>
<section class="chapter">
 <h2>It Continues: The Second Chapter</h2>
 <p>Bla dee bla, dee bla dee bla. Boom.</p>
</section>
<section class="chapter">
 <h2>Chapter Three: A Further Example</h2>
 <p>It's not like a battle between brightness and earthtones would go
 unnoticed.</p>
 <p>But it might ruin my story.</p>
</section>
<section class="appendix">
 <h2>Appendix A: Overview of Examples</h2>
 <p>These are demonstrations.</p>
</section>
<section class="appendix">
 <h2>Appendix B: Some Closing Remarks</h2>
 <p>Hopefully this long example shows that you <em>can</em> style
 sections, so long as they are used to indicate actual sections.</p>
</section>
```

4.3.4 The nav element §[p219]

Categories[p154]:

Flow content[p157].
Sectioning content[p157].
Palpable content[p158].

Contexts in which this element can be used[p154]:

Where sectioning content[p157] is expected.

Content model[p154]:

Flow content[p157].

Tag omission in text/html[p154]:

Neither tag is omissible.

Content attributes[p154]:

Global attributes[p162]

Accessibility considerations[p154]:

For authors.
For implementers.

Sanitization[p154]:

Default[p1249].

DOM interface[p154]:

Uses HTMLElement[p149].

The nav[p219] element represents[p148] 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[p219] 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[p227] element alone is sufficient for such cases; while a nav[p219] 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[p219] elements, one for primary navigation around the site, and one for secondary navigation around the page itself.

```
<body>
 <h1>The Wiki Center Of Exampland</h1>
 <nav>
  <ul>
   <li><a href="/">Home</a></li>
   <li><a href="/events">Current Events</a></li>
   ...more...
  </ul>
 </nav>
 <article>
  <header>
   <h2>Demos in Exampland</h2>
   <p>Written by A. N. Other.</p>
  </header>
  <nav>
```

```
   <ul>
    <li><a href="#public">Public demonstrations</a></li>
    <li><a href="#destroy">Demolitions</a></li>
    ...more...
   </ul>
  </nav>
  <div>
   <section id="public">
    <h2>Public demonstrations</h2>
    <p>...more...</p>
   </section>
   <section id="destroy">
    <h2>Demolitions</h2>
    <p>...more...</p>
   </section>
   ...more...
  </div>
  <footer>
   <p><a href="?edit">Edit</a> | <a href="?delete">Delete</a> | <a href="?Rename">Rename</a></p>
  </footer>
 </article>
 <footer>
  <p><small>© copyright 1998 Exampland Emperor</small></p>
 </footer>
</body>
```

Example

In the following example, the page has several places where links are present, but only one of those places is considered a navigation section.

```
<body itemscope itemtype="http://schema.org/Blog">
 <header>
  <h1>Wake up sheeple!</h1>
  <p><a href="news.html">News</a> -
     <a href="blog.html">Blog</a> -
     <a href="forums.html">Forums</a></p>
  <p>Last Modified: <span itemprop="dateModified">2009-04-01</span></p>
  <nav>
   <h2>Navigation</h2>
   <ul>
    <li><a href="articles.html">Index of all articles</a></li>
    <li><a href="today.html">Things sheeple need to wake up for today</a></li>
    <li><a href="successes.html">Sheeple we have managed to wake</a></li>
   </ul>
  </nav>
 </header>
 <main>
  <article itemprop="blogPosts" itemscope itemtype="http://schema.org/BlogPosting">
   <header>
    <h2 itemprop="headline">My Day at the Beach</h2>
   </header>
   <div itemprop="articleBody">
    <p>Today I went to the beach and had a lot of fun.</p>
    ...more content...
   </div>
   <footer>
    <p>Posted <time itemprop="datePublished" datetime="2009-10-10">Thursday</time>.</p>
   </footer>
  </article>
```

```
  ...more blog posts...
 </main>
 <footer>
  <p>Copyright ©
   <span itemprop="copyrightYear">2010</span>
   <span itemprop="copyrightHolder">The Example Company</span>
  </p>
  <p><a href="about.html">About</a> -
     <a href="policy.html">Privacy Policy</a> -
     <a href="contact.html">Contact Us</a></p>
 </footer>
</body>
```

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[p219] element doesn't have to contain a list, it can contain other kinds of content as well. In this navigation block, links are provided in prose:

```
<nav>
 <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[p219] is used in an email application, to let the user switch folders:

```
<p><input type=button value="Compose" onclick="compose()"></p>
<nav>
 <h1>Folders</h1>
 <ul>
  <li> <a href="/inbox" onclick="return openFolder(this.href)">Inbox</a> <span class=count></span>
  <li> <a href="/sent" onclick="return openFolder(this.href)">Sent</a>
  <li> <a href="/drafts" onclick="return openFolder(this.href)">Drafts</a>
  <li> <a href="/trash" onclick="return openFolder(this.href)">Trash</a>
  <li> <a href="/customers" onclick="return openFolder(this.href)">Customers</a>
 </ul>
</nav>
```

4.3.5 The aside element §[p222]

Categories[p154]:

Flow content[p157].
Sectioning content[p157].
Palpable content[p158].

Contexts in which this element can be used[p154]:

Where sectioning content[p157] is expected.

Content model[p154]:

Flow content[p157].

Tag omission in text/html[p154]:

Neither tag is omissible.

Content attributes[p154]:

Global attributes[p162]

Accessibility considerations[p154]:

For authors.
For implementers.

Sanitization[p154]:

Default[p1249].

DOM interface[p154]:

Uses HTMLElement[p149].

The aside[p222] element represents[p148] a section of a page that consists of content that is tangentially related to the content around the aside[p222] 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[p219] 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[p222] 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.

```
<aside>
 <h2>Switzerland</h2>
 <p>Switzerland, a land-locked country in the middle of geographic
 Europe, has not joined the geopolitical European Union, though it is
 a signatory to a number of European treaties.</p>
</aside>
```

Example

The following example shows how an aside is used to mark up a pull quote in a longer article.

```
...

<p>He later joined a large company, continuing on the same work.
<q>I love my job. People ask me what I do for fun when I'm not at
work. But I'm paid to do my hobby, so I never know what to
answer. Some people wonder what they would do if they didn't have to
work... but I know what I would do, because I was unemployed for a
year, and I filled that time doing exactly what I do now.</q></p>
```

```
<aside>
 <q>People ask me what I do for fun when I'm not at work. But I'm
 paid to do my hobby, so I never know what to answer.</q>
</aside>

<p>Of course his work — or should that be hobby? —
isn't his only passion. He also enjoys other pleasures.</p>

...
```

Example

The following extract shows how aside[p222] can be used for blogrolls and other side content on a blog:

```
<body>
 <header>
  <h1>My wonderful blog</h1>
  <p>My tagline</p>
 </header>
 <aside>
  <!-- this aside contains two sections that are tangentially related
  to the page, namely, links to other blogs, and links to blog posts
  from this blog -->
  <nav>
   <h2>My blogroll</h2>
   <ul>
    <li><a href="https://blog.example.com/">Example Blog</a>
   </ul>
  </nav>
  <nav>
   <h2>Archives</h2>
   <ol reversed>
    <li><a href="/last-post">My last post</a>
    <li><a href="/first-post">My first post</a>
   </ol>
  </nav>
 </aside>
 <aside>
  <!-- this aside is tangentially related to the page also, it
  contains twitter messages from the blog author -->
  <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 -->
  <h2>My last post</h2>
  <p>This is my last post.</p>
  <footer>
   <p><a href="/last-post" rel=bookmark>Permalink</a>
  </footer>
 </article>
 <article>
  <!-- this is also a blog post -->
  <h2>My first post</h2>
```

```
  <p>This is my first post.</p>
  <aside>
   <!-- this aside is about the blog post, since it's inside the
   <article> element; it would be wrong, for instance, to put the
   blogroll here, since the blogroll isn't really related to this post
   specifically, only to the page as a whole -->
   <h2>Posting</h2>
   <p>While I'm thinking about it, I wanted to say something about
   posting. Posting is fun!</p>
  </aside>
  <footer>
   <p><a href="/first-post" rel=bookmark>Permalink</a>
  </footer>
 </article>
 <footer>
  <p><a href="/archives">Archives</a> -
   <a href="/about">About me</a> -
   <a href="/copyright">Copyright</a></p>
 </footer>
</body>
```

4.3.6 The h1, h2, h3, h4, h5, and h6 elements §[p22] 4

Categories[p154]:

Flow content[p157].
Heading content[p157].
Palpable content[p158].

Contexts in which this element can be used[p154]:

As a child of an hgroup[p225] element.
Where heading content[p157] is expected.

Content model[p154]:

Phrasing content[p157].

Tag omission in text/html[p154]:

Neither tag is omissible.

Content attributes[p154]:

Global attributes[p162]

Accessibility considerations[p154]:

For authors.
For implementers.

Sanitization[p154]:

Default[p1249].

DOM interface[p154]:

IDL

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

  // also has obsolete members
};
```

These elements represent[p148] headings for their sections.

The semantics and meaning of these elements are defined in the section on headings and outlines[p231].

These elements have a heading level[p231] given by the number in their name. The heading level[p231] corresponds to the levels of nested sections. The h1[p224] element is for a top-level section, h2[p224] for a subsection, h3[p224] for a sub-subsection, and so on.

Example

As far as their respective document outlines (their heading and section structures) are concerned, these two snippets are semantically equivalent:

```
<body>
<h1>Let's call it a draw(ing surface)</h1>
<h2>Diving in</h2>
<h2>Simple shapes</h2>
<h2>Canvas coordinates</h2>
<h3>Canvas coordinates diagram</h3>
<h2>Paths</h2>
</body>
```

```
<body>
 <h1>Let's call it a draw(ing surface)</h1>
 <section>
  <h2>Diving in</h2>
 </section>
 <section>
  <h2>Simple shapes</h2>
 </section>
 <section>
  <h2>Canvas coordinates</h2>
  <section>
   <h3>Canvas coordinates diagram</h3>
  </section>
 </section>
 <section>
  <h2>Paths</h2>
 </section>
</body>
```

Authors might prefer the former style for its terseness, or the latter style for its additional styling hooks. Which is best is purely an issue of preferred authoring style.

4.3.7 The hgroup element §[p225]

Categories[p154]:

Flow content[p157].
Heading content[p157].
Palpable content[p158].

Contexts in which this element can be used[p154]:

Where heading content[p157] is expected.

Content model[p154]:

Zero or more p[p238] elements, followed by one h1[p224], h2[p224], h3[p224], h4[p224], h5[p224], or h6[p224] element, followed by zero or more p[p238] elements, optionally intermixed with script-supporting elements[p159].

Tag omission in text/html[p154]:

Neither tag is omissible.

Content attributes[p154]:

Global attributes[p162]

Accessibility considerations[p154]:

For authors.

For implementers.

Sanitization[p154]:

Default[p1249].

DOM interface[p154]:

Uses HTMLElement[p149].

The hgroup[p225] element represents[p148] a heading and related content. The element may be used to group an h1[p224]–h6[p224] element with one or more p[p238] elements containing content representing a subheading, alternative title, or tagline.

Example

Here are some examples of valid headings contained within an hgroup[p225] element.

```
<hgroup>
 <h1>The reality dysfunction</h1>
 <p>Space is not the only void</p>
</hgroup>
```

```
<hgroup>
 <h1>Dr. Strangelove</h1>
 <p>Or: How I Learned to Stop Worrying and Love the Bomb</p>
</hgroup>
```

MDN

4.3.8 The header element §[p226]

Categories[p154]:

Flow content[p157].
Palpable content[p158].

Contexts in which this element can be used[p154]:

Where flow content[p157] is expected.

Content model[p154]:

Flow content[p157], but with no header[p226] or footer[p227] element descendants.

Tag omission in text/html[p154]:

Neither tag is omissible.

Content attributes[p154]:

Global attributes[p162]

Accessibility considerations[p154]:

If there is an ancestor sectioning content[p157] element: for authors; for implementers.
Otherwise: for authors; for implementers.

Sanitization[p154]:

Default[p1249].

DOM interface[p154]:

Uses HTMLElement[p149].

The header[p226] element represents[p148] a group of introductory or navigational aids.

Note

A header[p226] element is intended to usually contain a heading (an h1[p224]–h6[p224] element or an hgroup[p225] element), but this is not required. The header[p226] element can also be used to wrap a section's table of contents, a search form, or any relevant logos.

Example

Here are some sample headers. This first one is for a game:

```
<header>
 <p>Welcome to...</p>
 <h1>Voidwars!</h1>
</header>
```

The following snippet shows how the element can be used to mark up a specification's header:

```
<header>
 <hgroup>
  <h1>Fullscreen API</h1>
  <p>Living Standard — Last Updated 19 October 2015<p>
 </hgroup>
 <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[p226] element is not sectioning content[p157]; it doesn't introduce a new section.

Example

In this example, the page has a page heading given by the h1[p224] element, and two subsections whose headings are given by h2[p224] elements. The content after the header[p226] element is still part of the last subsection started in the header[p226] element, because the header[p226] element doesn't take part in the outline[p231] algorithm.

```
<body>
 <header>
  <h1>Little Green Guys With Guns</h1>
  <nav>
   <ul>
    <li><a href="/games">Games</a>
    <li><a href="/forum">Forum</a>
    <li><a href="/download">Download</a>
   </ul>
  </nav>
  <h2>Important News</h2> <!-- this starts a second subsection -->
  <!-- this is part of the subsection entitled "Important News" -->
  <p>To play today's games you will need to update your client.</p>
  <h2>Games</h2> <!-- this starts a third subsection -->
 </header>
 <p>You have three active games:</p>
 <!-- this is still part of the subsection entitled "Games" -->
 ...
```

4.3.9 The footer element §[p227]

Categories[p154]:

Flow content[p157].
Palpable content[p158].

Contexts in which this element can be used[p154]:

Where flow content[p157] is expected.

Content model[p154]:

Flow content[p157], but with no `header`[p226] or `footer`[p227] element descendants.

Tag omission in text/html[p154]:

Neither tag is omissible.

Content attributes[p154]:

Global attributes[p162]

Accessibility considerations[p154]:

If there is an ancestor sectioning content[p157] element: for authors; for implementers.
Otherwise: for authors; for implementers.

Sanitization[p154]:

Default[p1249].

DOM interface[p154]:

Uses `HTMLElement`[p149].

The `footer`[p227] element represents[p148] a footer for its nearest ancestor sectioning content[p157] element, or for the body element[p144] if there is no such ancestor. A footer typically contains information about its section such as who wrote it, links to related documents, copyright data, and the like.

When the `footer`[p227] element contains entire sections, they represent[p148] appendices, indices, long colophons, verbose license agreements, and other such content.

Note

Contact information for the author or editor of a section belongs in an `address`[p230] element, possibly itself inside a `footer`[p227]. Bylines and other information that could be suitable for both a `header`[p226] or a `footer`[p227] 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 there is no ancestor sectioning content[p157] element, then it applies to the whole page.

Note

The `footer`[p227] element is not itself sectioning content[p157]; 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>
  <p>The ipsum of all lorems</p>
 </hgroup>
 <p>A dolor sit amet, consectetur adipisicing elit, sed do eiusmod
 tempor incididunt ut labore et dolore magna aliqua. Ut enim ad minim
 veniam, quis nostrud exercitation ullamco laboris nisi ut aliquip ex
 ea commodo consequat. Duis aute irure dolor in reprehenderit in
 voluptate velit esse cillum dolore eu fugiat nulla
 pariatur. Excepteur sint occaecat cupidatat non proident, sunt in
 culpa qui officia deserunt mollit anim id est laborum.</p>
 <footer><a href="../">Back to index...</a></footer>
</body>
```

Example

Here is an example which shows the footer[p227] element being used both for a site-wide footer and for a section footer.

```
<!DOCTYPE HTML>
<HTML LANG="en"><HEAD>
<TITLE>The Ramblings of a Scientist</TITLE>
<BODY>
<H1>The Ramblings of a Scientist</H1>
<ARTICLE>
 <H1>Episode 15</H1>
 <VIDEO SRC="/fm/015.ogv" CONTROLS PRELOAD>
  <P><A HREF="/fm/015.ogv">Download video</A>.</P>
 </VIDEO>
 <FOOTER> <!-- footer for article -->
  <P>Published <TIME DATETIME="2009-10-21T18:26-07:00">on 2009/10/21 at 6:26pm</TIME></P>
 </FOOTER>
</ARTICLE>
<ARTICLE>
 <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> <!-- footer for article -->
  <P>Published <TIME DATETIME="2009-09-15T14:54-07:00">on 2009/09/15 at 2:54pm</TIME></P>
 </FOOTER>
</ARTICLE>
<FOOTER> <!-- site wide footer -->
 <NAV>
  <P><A HREF="/credits.html">Credits</A> —
     <A HREF="/tos.html">Terms of Service</A> —
     <A HREF="/index.html">Blog Index</A></P>
 </NAV>
 <P>Copyright © 2009 Gordon Freeman</P>
</FOOTER>
</BODY>
</HTML>
```

Example

Some site designs have what is sometimes referred to as "fat footers" — footers that contain a lot of material, including images, links to other articles, links to pages for sending feedback, special offers... in some ways, a whole "front page" in the footer.

This fragment shows the bottom of a page on a site with a "fat footer":

```
...
 <footer>
  <nav>
   <section>
    <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 clif<!-- sic -->? Our guest writer Lara shows you how to bumble
    your way through the bars. <a href="articles/kindplus/1">Read
    more...</a></p>
    <p><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>
   <ul>
```

```
    <li> <a href="/about">About us...</a>
    <li> <a href="/feedback">Send feedback!</a>
    <li> <a href="/sitemap">Sitemap</a>
   </ul>
  </nav>
  <p><small>Copyright © 2015 The Snacker —
  <a href="/tos">Terms of Service</a></small></p>
 </footer>
</body>
```

4.3.10 The address element §[p230]

Categories[p154]:

Flow content[p157].
Palpable content[p158].

Contexts in which this element can be used[p154]:

Where flow content[p157] is expected.

Content model[p154]:

Flow content[p157], but with no heading content[p157] descendants, no sectioning content[p157] descendants, and no header[p226], footer[p227], or address[p230] element descendants.

Tag omission in text/html[p154]:

Neither tag is omissible.

Content attributes[p154]:

Global attributes[p162]

Accessibility considerations[p154]:

For authors.
For implementers.

Sanitization[p154]:

Default[p1249].

DOM interface[p154]:

Uses HTMLElement[p149].

The address[p230] element represents[p148] the contact information for its nearest article[p214] or body[p212] element ancestor. If that is the body element[p144], 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:

```
<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[p230] 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[p238] element is the appropriate element for marking up postal addresses in general.)

The address[p230] element must not contain information other than contact information.

Example

For example, the following is non-conforming use of the address element:

```
<ADDRESS>Last Modified: 1999/12/24 23:37:50</ADDRESS>
```

Typically, the address element would be included along with other information in a footer element.

The contact information for a node *node* is a collection of address elements defined by the first applicable entry from the following list:

↪ **If *node* is an article element**

↪ **If *node* is a body element**

The contact information consists of all the address elements that have *node* as an ancestor and do not have another body or article element ancestor that is a descendant of *node*.

↪ **If *node* has an ancestor element that is an article element**

↪ **If *node* has an ancestor element that is a body element**

The contact information of *node* is the same as the contact information of the nearest article or body element ancestor, whichever is nearest.

↪ **If *node*'s node document has a body element**

The contact information of *node* is the same as the contact information of the body element of the Document.

↪ **Otherwise**

There is no contact information for *node*.

User agents may expose the contact information of a node to the user, or use it for other purposes, such as indexing sections based on the sections' contact information.

Example

In this example the footer contains contact information and a copyright notice.

```
<footer>
 <address>
  For more details, contact
  <a href="mailto:js@example.com">John Smith</a>.
 </address>
 <p><small>© copyright 2038 Example Corp.</small></p>
</footer>
```

4.3.11 Headings and outlines §1

h1–h6 elements have a **heading level**, which is given by getting the element's computed heading level.

These elements represent **headings**. The lower a heading's heading level is, the fewer ancestor sections the heading has.

The **outline** is all headings in a document, in tree order.

The outline should be used for generating document outlines, for example when generating tables of contents. When creating an interactive table of contents, entries should jump the user to the relevant heading.

If a document has one or more headings, at least a single heading within the outline should have a heading level of 1.

Each heading following another heading *lead* in the outline must have a heading level that is less than, equal to, or 1 greater than *lead*'s heading level.

Example

The following example is non-conforming:

```
<body>
 <h1>Apples</h1>
 <p>Apples are fruit.</p>
 <section>
  <h3>Taste</h3>
  <p>They taste lovely.</p>
 </section>
</body>
```

It could be written as follows and then it would be conforming:

```
<body>
 <h1>Apples</h1>
 <p>Apples are fruit.</p>
 <section>
  <h2>Taste</h2>
  <p>They taste lovely.</p>
 </section>
</body>
```

4.3.11.1 Heading levels & offsets §[p232]

The **headingoffset** content attribute allows authors to offset heading levels for descendants.

If the headingoffset[p232] attribute is specified, it must have a value that is a valid non-negative integer[p81] between 0 and 8, inclusive.

The **headingreset** content attribute is a boolean attribute[p79]. It allows authors to prevent a heading offset computation from traversing beyond the element with the attribute.

To **get an element's computed heading level**, given an element *element*:

1. Let *level* be 0.
2. If *element*'s local name is h1[p224], then set *level* to 1.
3. If *element*'s local name is h2[p224], then set *level* to 2.
4. If *element*'s local name is h3[p224], then set *level* to 3.
5. If *element*'s local name is h4[p224], then set *level* to 4.
6. If *element*'s local name is h5[p224], then set *level* to 5.
7. If *element*'s local name is h6[p224], then set *level* to 6.
8. Assert: *level* is not 0.
9. Increment *level* by the result of getting an element's computed heading offset[p232] given *element*.
10. If *level* is greater than 9, then return 9.
11. Return *level*.

To **get an element's computed heading offset**, given an element *element*, perform the following steps. They return a non-negative integer.

1. Let *offset* be 0.
2. Let *inclusiveAncestor* be *element*.
3. While *inclusiveAncestor* is not null:

1. Let *nextOffset* be 0.
2. If *inclusiveAncestor* is an HTML element[p46] and has a `headingoffset`[p232] attribute, then parse its value using the rules for parsing non-negative integers[p81].

   If the result of parsing the value is not an error, then set *nextOffset* to that value.
3. Increment *offset* by *nextOffset*.
4. If *inclusiveAncestor* is an HTML element[p46] and has a `headingreset`[p232] attribute, then return *offset*.
5. If *inclusiveAncestor*'s parent is a shadow root, then set *inclusiveAncestor* to that shadow root's host and continue.
6. Set *inclusiveAncestor* to *inclusiveAncestor*'s parent element.

4. Return *offset*.

Example

This example shows a combination of `headingoffset`[p232], `headingreset`[p232], and `aria-level` attributes with comments demonstrating the respective heading levels. This example illustrates the various combinations and is not a best practice example.

```
<body>
  <main>
   <h1>This is a heading level 1</h1>
   <article headingoffset="1">
    <h1>This is a heading level 2</h1>
    <section headingoffset="1">
     <h1>This is a heading level 3</h1>
     <dialog headingreset>
      <h1>This is a heading level 1</h1>
     </dialog>
    </section>
   </article>
   <h1 aria-level="2">This is a heading level 2</h1>
  </main>
</body>
```

4.3.11.2 Sample outlines §[p233]

Example

The following markup fragment:

```
<body>
  <hgroup id="document-title">
    <h1>HTML: Living Standard</h1>
    <p>Last Updated 12 August 2016</p>
  </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 3 document headings:

1. `<h1>HTML: Living Standard</h1>`
2. `<h2>Table of contents</h2>`.
3. `<h2>First section</h2>`.

A rendered view of the outline[p231] might look like:

HTML: Living Standard

Table of contents

First section

Example

First, here is a 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>
<p>Earning money is good.</p>
<h3>Getting a job</h3>
<p>To earn money you typically need a job.</p>
<h2>Spending money</h2>
<p>Spending is what money is mainly used for.</p>
<h3>Cheap things</h3>
<p>Buying cheap things often not cost-effective.</p>
<h3>Expensive things</h3>
<p>The most expensive thing is often not the most cost-effective either.</p>
<h2>Investing money</h2>
<p>You can lend your money to other people.</p>
<h2>Losing money</h2>
<p>If you spend money or invest money, sooner or later you will lose money.
<h3>Poor judgement</h3>
<p>Usually if you lose money it's because you made a mistake.</p>
```

Its outline[p231] could be presented as follows:

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

Notice that the `title`[p181] element is not a heading[p231].

Example

A document can contain multiple top-level headings:

```
<!DOCTYPE HTML>
<html lang=en>
<title>Alphabetic Fruit</title>
<h1>Apples</h1>
<p>Pomaceous.</p>
<h1>Bananas</h1>
<p>Edible.</p>
<h1>Carambola</h1>
<p>Star.</p>
```

The document's outline[p231] could be presented as follows:

1. Apples
2. Bananas
3. Carambola

Example

header[p226] elements do not influence the outline[p231] of a document:

```
<!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>
  <h2>We're adopting a child!</h2>
 </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 document's outline[p231] could be presented as follows:

1. Ray's blog
    1. We're adopting a child!

Example

The following example is conforming, but not encouraged as it has no heading[p231] whose heading level[p231] is 1:

```
<!DOCTYPE HTML>
<html lang=en>
<title>Alphabetic Fruit</title>
<section>
 <h2>Apples</h2>
 <p>Pomaceous.</p>
</section>
<section>
 <h2>Bananas</h2>
 <p>Edible.</p>
</section>
<section>
 <h2>Carambola</h2>
 <p>Star.</p>
</section>
```

The document's outline[p231] could be presented as follows:

1.
    1. Apples
    2. Bananas
    3. Carambola

Example

The following example is conforming, but not encouraged as the first heading[p231]'s heading level[p231] is not 1:

```
<!DOCTYPE HTML>
<html lang=en>
<title>Feathers on The Site of Encyclopedic Knowledge</title>
 <h2>A plea from our caretakers</h2>
 <p>Please, we beg of you, send help! We're stuck in the server room!</p>
<h1>Feathers</h1>
<p>Epidermal growths.</p>
```

The document's outline[p231] could be presented as follows:

1. 1. A plea from our caretakers
2. Feathers

4.3.11.3 Exposing outlines to users §[p236]

User agents are encouraged to expose page outlines[p231] to users to aid in navigation. This is especially true for non-visual media, e.g. screen readers.

Example

For instance, a user agent could map the arrow keys as follows:

Shift + ← Left
Go to previous heading

Shift + → Right
Go to next heading

Shift + ↑ Up
Go to next heading whose level[p231] is one less than the current heading's level

Shift + ↓ Down
Go to next heading whose level[p231] is the same as the current heading's level

4.3.12 Usage summary §[p236]

This section is non-normative.

Element	Purpose Example

body[p212]	The contents of the document.
	`<!DOCTYPE HTML>`
	`<html lang="en">`
	`<head> <title>Steve Hill's Home Page</title> </head>`
	`<body> <p>Hard Trance is My Life.</p> </body>`
	`</html>`

article[p214]	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.
	`<article>`
	`<img src="/tumblr_masqy2s5yn1rzfqbpo1_500.jpg" alt="Yellow smiley face with the caption 'masif'">`
	`<p>My fave Masif tee so far!</p>`
	`<footer>Posted 2 days ago</footer>`
	`</article>`
	`<article>`
	`<img src="/tumblr_m9tf6wSr6W1rzfqbpo1_500.jpg" alt="">`
	`<p>Happy 2nd birthday Masif Saturdays!!!</p>`

Element	Purpose Example
	```<footer>Posted 3 weeks ago</footer>```
	```</article>```
section[p216]	A generic section of a document or application. A section, in this context, is a thematic grouping of content, typically with a heading. ``` Biography ```
	```<section>```
	``` The facts ```
	``` 1500+ shows, 14+ countries ```
	```</section>```
	```<section>```
	``` 2010/2011 figures per year ```
	``` 100+ shows, 8+ countries ```
	```</section>```
nav[p219]	A section of a page that links to other pages or to parts within the page: a section with navigation links. ```<nav>```
	```<p>Home```
	```<p>Bio```
	```<p>Discog```
	```</nav>```
aside[p222]	A section of a page that consists of content that is tangentially related to the content around the aside[p222] element, and which could be considered separate from that content. Such sections are often represented as sidebars in printed typography. ``` Music ```
	``` As any burner can tell you, the event has a lot of trance. ```
	```<aside>You can buy the music we played at our playlist page.</aside>```
	``` This year we played a kind of trance that originated in Belgium, Germany, and the Netherlands in the mid-90s. ```
h1[p224]–h6[p224]	A heading ``` The Guide To Music On The Playa ```
	``` The Main Stage ```
	``` If you want to play on a stage, you should bring one. ```
	``` Amplified Music ```
	``` Amplifiers up to 300W or 90dB are welcome. ```
hgroup[p225]	A heading and related content. The element may be used to group an h1[p224]–h6[p224] element with one or more p[p238] elements containing content representing a subheading, alternative title, or tagline. ```<hgroup>```
	``` Burning Music ```
	``` The Guide To Music On The Playa ```
	```</hgroup>```
	```<section>```
	```<hgroup>```
	``` Main Stage ```
	``` The Fiction Of A Music Festival ```
	```</hgroup>```
	``` If you want to play on a stage, you should bring one. ```
	```</section>```
	```<section>```
	```<hgroup>```
	``` Loudness! ```
	``` Questions About Amplified Music ```
	```</hgroup>```
	``` Amplifiers up to 300W or 90dB are welcome. ```
	```</section>```
header[p226]	A group of introductory or navigational aids. ```<article>```
	```<header>```
	``` Hard Trance is My Life ```
	``` By DJ Steve Hill and Technikal ```
	```</header>```
	``` The album with the amusing punctuation has red artwork. ```
	```</article>```
footer[p227]	A footer for its nearest ancestor sectioning content[p157] element, or for the body element[p144] if there is no such ancestor. A footer typically contains information about its section such as who wrote it, links to related documents, copyright data, and the like. ```<article>```
	``` Hard Trance is My Life ```
	``` The album with the amusing punctuation has red artwork. ```

Element	Purpose Example
	<footer> Artists: DJ Steve Hill and Technikal </footer> </article>

4.3.12.1 Article or section? §[p238]

This section is non-normative.

A section[p216] forms part of something else. An article[p214] 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[p216] 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 article[p214]s because that was the whole thing.

A comment on an article is not part of the article[p214] on which it is commenting, therefore it is its own article[p214].

4.4 Grouping content §[p238]

4.4.1 The p element §[p238]

Categories[p154]:

Flow content[p157].
Palpable content[p158].

Contexts in which this element can be used[p154]:

Where flow content[p157] is expected.
As a child of an hgroup[p225] element.

Content model[p154]:

Phrasing content[p157].

Tag omission in text/html[p154]:

A p[p238] element's end tag[p1359] can be omitted if the p[p238] element is immediately followed by an address[p230], article[p214], aside[p222], blockquote[p244], details[p666], dialog[p675], div[p266], dl[p253], fieldset[p620], figcaption[p262], figure[p259], footer[p227], form[p532], h1[p224], h2[p224], h3[p224], h4[p224], h5[p224], h6[p224], header[p226], hgroup[p225], hr[p241], main[p262], menu[p250], nav[p219], ol[p247], p[p238], pre[p242], search[p264], section[p216], table[p496], or ul[p249] element, or if there is no more content in the parent element and the parent element is an HTML element[p46] that is not an a[p267], audio[p425], del[p351], ins[p350], map[p487], noscript[p703], or video[p420] element, or an autonomous custom element[p796].

Content attributes[p154]:

Global attributes[p162]

Accessibility considerations[p154]:

For authors.
For implementers.

Sanitization[p154]:

Default[p1249].

DOM interface[p154]:

IDL
```
[Exposed=Window]
interface HTMLParagraphElement : HTMLElement {
  [HTMLConstructor] constructor();
```

```
  // also has obsolete members
};
```

The p[p238] element represents[p148] a paragraph[p159].

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:

```
<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,<br>
Whose markup was of dubious quality.<br>
The validator complained,<br>
So the author was pained,<br>
To move the error from the markup to the rhyming.</p>
```

The p[p238] element should not be used when a more specific element is more appropriate.

Example

The following example is technically correct:

```
<section>
 <!-- ... -->
 <p>Last modified: 2001-04-23</p>
 <p>Author: fred@example.com</p>
</section>
```

However, it would be better marked-up as:

```
<section>
 <!-- ... -->
 <footer>Last modified: 2001-04-23</footer>
 <address>Author: fred@example.com</address>
</section>
```

Or:

```
<section>
 <!-- ... -->
 <footer>
```

```
  <p>Last modified: 2001-04-23</p>
  <address>Author: fred@example.com</address>
 </footer>
</section>
```

Note

List elements (in particular, ol[p247] and ul[p249] elements) cannot be children of p[p238] 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[p159], in HTML terms, is not a logical concept, but a structural one. In the fantastic example above, there are actually five paragraphs[p159] 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:

```
<p>For instance, this fantastic sentence has bullets relating to</p>
<ul>
 <li>wizards,
 <li>faster-than-light travel, and
 <li>telepathy,
</ul>
<p>and is further discussed below.</p>
```

Authors wishing to conveniently style such "logical" paragraphs consisting of multiple "structural" paragraphs can use the div[p266] element instead of the p[p238] element.

Example

Thus for instance the above example could become the following:

```
<div>For instance, this fantastic sentence has bullets relating to
<ul>
 <li>wizards,
 <li>faster-than-light travel, and
 <li>telepathy,
</ul>
and is further discussed below.</div>
```

This example still has five structural paragraphs, but now the author can style just the div[p266] instead of having to consider each part of the example separately.

4.4.2 The hr element §[p241]

Categories[p154]:

Flow content[p157].

Contexts in which this element can be used[p154]:

Where flow content[p157] is expected.
As a descendant of a select[p590] element.

Content model[p154]:

Nothing[p155].

Tag omission in text/html[p154]:

No end tag[p1359].

Content attributes[p154]:

Global attributes[p162]

Accessibility considerations[p154]:

For authors.
For implementers.

Sanitization[p154]:

Default[p1249].

DOM interface[p154]:

IDL

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

  // also has obsolete members
};
```

The hr[p241] element represents[p148] a paragraph[p159]-level thematic break, e.g., a scene change in a story, or a transition to another topic within a section of a reference book; alternatively, it represents a separator between a set of options of a select[p590] element.

Example

The following fictional extract from a project manual shows two sections that use the hr[p241] element to separate topics within the section.

```
<section>
 <h1>Communication</h1>
 <p>There are various methods of communication. This section
 covers a few of the important ones used by the project.</p>
 <hr>
 <p>Communication stones seem to come in pairs and have mysterious
 properties:</p>
 <ul>
  <li>They can transfer thoughts in two directions once activated
  if used alone.</li>
  <li>If used with another device, they can transfer one's
  consciousness to another body.</li>
  <li>If both stones are used with another device, the
  consciousnesses switch bodies.</li>
 </ul>
 <hr>
 <p>Radios use the electromagnetic spectrum in the meter range and
 longer.</p>
 <hr>
 <p>Signal flares use the electromagnetic spectrum in the
 nanometer range.</p>
</section>
```

```
<section>
 <h1>Food</h1>
 <p>All food at the project is rationed:</p>
 <dl>
  <dt>Potatoes</dt>
  <dd>Two per day</dd>
  <dt>Soup</dt>
  <dd>One bowl per day</dd>
 </dl>
 <hr>
 <p>Cooking is done by the chefs on a set rotation.</p>
</section>
```

There is no need for an hr[p241] element between the sections themselves, since the section[p216] elements and the h1[p224] elements imply thematic changes themselves.

Example

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[p241] element.

```
<p>Dudley was ninety-two, in his second life, and fast approaching
time for another rejuvenation. Despite his body having the physical
age of a standard fifty-year-old, the prospect of a long degrading
campaign within academia was one he regarded with dread. For a
supposedly advanced civilization, the Intersolar Commonwealth could be
appallingly backward at times, not to mention cruel.</p>
<p><i>Maybe it won't be that bad</i>, he told himself. The lie was
comforting enough to get him through the rest of the night's
shift.</p>
<hr>
<p>The Carlton AllLander drove Dudley home just after dawn. Like the
astronomer, the vehicle was old and worn, but perfectly capable of
doing its job. It had a cheap diesel engine, common enough on a
semi-frontier world like Gralmond, although its drive array was a
thoroughly modern photoneural processor. With its high suspension and
deep-tread tyres it could plough along the dirt track to the
observatory in all weather and seasons, including the metre-deep snow
of Gralmond's winters.</p>
```

Note

The hr[p241] element does not affect the document's outline[p231].

4.4.3 The pre element §[p242]

Categories[p154]:

Flow content[p157].
Palpable content[p158].

Contexts in which this element can be used[p154]:

Where flow content[p157] is expected.

Content model[p154]:

Phrasing content[p157].

Tag omission in text/html[p154]:

Neither tag is omissible.

Content attributes[p154]:

Global attributes[p162]

Accessibility considerations[p154]:

For authors.
For implementers.

Sanitization[p154]:

Default[p1249].

DOM interface[p154]:

IDL

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

  // also has obsolete members
};
```

The pre[p242] element represents[p148] a block of preformatted text, in which structure is represented by typographic conventions rather than by elements.

Note

In the HTML syntax[p1356], a leading newline character immediately following the pre[p242] element start tag is stripped.

Some examples of cases where the pre[p242] 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.

Note

Authors are encouraged to consider how preformatted text will be experienced when the formatting is lost, as will be the case for users of speech synthesizers, braille displays, and the like. For cases like ASCII art, it is likely that an alternative presentation, such as a textual description, would be more universally accessible to the readers of the document.

To represent a block of computer code, the pre[p242] element can be used with a code[p297] element; to represent a block of computer output the pre[p242] element can be used with a samp[p299] element. Similarly, the kbd[p300] element can be used within a pre[p242] element to indicate text that the user is to enter.

Note

This element has rendering requirements involving the bidirectional algorithm[p177].

Example

In the following snippet, a sample of computer code is presented.

```
<p>This is the <code>Panel</code> constructor:</p>
<pre><code>function Panel(element, canClose, closeHandler) {
  this.element = element;
  this.canClose = canClose;
  this.closeHandler = function () { if (closeHandler) closeHandler() };
}</code></pre>
```

Example

In the following snippet, samp[p299] and kbd[p300] elements are mixed in the contents of a pre[p242] element to show a session of Zork I.

```
<pre><samp>You are in an open field west of a big white house with a boarded
```

```
front door.
There is a small mailbox here.

></samp> <kbd>open mailbox</kbd>

<samp>Opening the mailbox reveals:
A leaflet.

></samp></pre>
```

Example

The following shows a contemporary poem that uses the pre[p242] element to preserve its unusual formatting, which forms an intrinsic part of the poem itself.

```
<pre>                maxling

it is with a          heart
               heavy

that i admit loss of a feline
        so           loved

a friend lost to the
        unknown
                                (night)

~cdr 11dec07</pre>
```

4.4.4 The blockquote element §[p244]

Categories[p154]:

Flow content[p157].
Palpable content[p158].

Contexts in which this element can be used[p154]:

Where flow content[p157] is expected.

Content model[p154]:

Flow content[p157].

Tag omission in text/html[p154]:

Neither tag is omissible.

Content attributes[p154]:

Global attributes[p162]
cite[p245] — Link to the source of the quotation or more information about the edit

Accessibility considerations[p154]:

For authors.
For implementers.

Sanitization[p154]:

Default[p1249] with attributes cite[p245].

DOM interface[p154]:

IDL

```
[Exposed=Window]
interface HTMLQuoteElement : HTMLElement {
```

```
  [HTMLConstructor] constructor();

  [CEReactions, ReflectURL] attribute USVString cite;
};
```

Note

The HTMLQuoteElement[p244] interface is also used by the q[p277] element.

The blockquote[p244] element represents[p148] a section that is quoted from another source.

Content inside a blockquote[p244] must be quoted from another source, whose address, if it has one, may be cited in the **cite** attribute.

If the cite[p245] attribute is present, it must be a valid URL potentially surrounded by spaces[p100]. To obtain the corresponding citation link, the value of the attribute must be parsed[p101] 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[p244] 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[p244] element.

Example

For example, here the attribution is given in a paragraph after the quote:

```
<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.

Example

Here a blockquote[p244] element is used in conjunction with a figure[p259] element and its figcaption[p262] to clearly relate a quote to its attribution (which is not part of the quote and therefore doesn't belong inside the blockquote[p244] itself):

```
<figure>
 <blockquote>
  <p>The truth may be puzzling. It may take some work to grapple with.
  It may be counterintuitive. It may contradict deeply held
  prejudices. It may not be consonant with what we desperately want to
  be true. But our preferences do not determine what's true. We have a
  method, and that method helps us to reach not absolute truth, only
  asymptotic approaches to the truth — never there, just closer
  and closer, always finding vast new oceans of undiscovered
  possibilities. Cleverly designed experiments are the key.</p>
 </blockquote>
 <figcaption>Carl Sagan, in "<cite>Wonder and Skepticism</cite>", from
```

```
 the <cite>Skeptical Inquirer</cite> Volume 19, Issue 1 (January-February
 1995)</figcaption>
</figure>
```

Example

This next example shows the use of cite[p275] alongside blockquote[p244]:

```
<p>His next piece was the aptly named <cite>Sonnet 130</cite>:</p>
<blockquote cite="https://quotes.example.org/s/sonnet130.html">
  <p>My mistress' eyes are nothing like the sun,<br>
  Coral is far more red, than her lips red,<br>
  ...
```

Example

This example shows how a forum post could use blockquote[p244] to show what post a user is replying to. The article[p214] element is used for each post, to mark up the threading.

```
<article>
 <h1><a href="https://bacon.example.com/?blog=109431">Bacon on a crowbar</a></h1>
 <article>
  <header><strong>t3yw</strong> 12 points 1 hour ago</header>
  <p>I bet a narwhal would love that.</p>
  <footer><a href="?pid=29578">permalink</a></footer>
  <article>
   <header><strong>greg</strong> 8 points 1 hour ago</header>
   <blockquote><p>I bet a narwhal would love that.</p></blockquote>
   <p>Dude narwhals don't eat bacon.</p>
   <footer><a href="?pid=29579">permalink</a></footer>
   <article>
    <header><strong>t3yw</strong> 15 points 1 hour ago</header>
    <blockquote>
     <blockquote><p>I bet a narwhal would love that.</p></blockquote>
     <p>Dude narwhals don't eat bacon.</p>
    </blockquote>
    <p>Next thing you'll be saying they don't get capes and wizard
    hats either!</p>
    <footer><a href="?pid=29580">permalink</a></footer>
    <article>
     <article>
      <header><strong>boing</strong> -5 points 1 hour ago</header>
      <p>narwhals are worse than ceiling cat</p>
      <footer><a href="?pid=29581">permalink</a></footer>
     </article>
    </article>
   </article>
  </article>
  <article>
   <header><strong>fred</strong> 1 points 23 minutes ago</header>
   <blockquote><p>I bet a narwhal would love that.</p></blockquote>
   <p>I bet they'd love to peel a banana too.</p>
   <footer><a href="?pid=29582">permalink</a></footer>
  </article>
 </article>
</article>
```

Example

This example shows the use of a blockquote[p244] for short snippets, demonstrating that one does not have to use p[p238] elements inside blockquote[p244] elements:

```
<p>He began his list of "lessons" with the following:</p>
<blockquote>One should never assume that his side of
the issue will be recognized, let alone that it will
be conceded to have merits.</blockquote>
<p>He continued with a number of similar points, ending with:</p>
<blockquote>Finally, one should be prepared for the threat
of breakdown in negotiations at any given moment and not
be cowed by the possibility.</blockquote>
<p>We shall now discuss these points...
```

Note

Examples of how to represent a conversation[p815] are shown in a later section; it is not appropriate to use the cite[p275] and blockquote[p244] elements for this purpose.

4.4.5 The ol element §[p247]

Categories[p154]:

Flow content[p157].

If the element's children include at least one li[p251] element: Palpable content[p158].

Contexts in which this element can be used[p154]:

Where flow content[p157] is expected.

Content model[p154]:

Zero or more li[p251] and script-supporting[p159] elements.

Tag omission in text/html[p154]:

Neither tag is omissible.

Content attributes[p154]:

Global attributes[p162]
reversed[p248] — Number the list backwards
start[p248] — Starting value[p248] of the list
type[p248] — Kind of list marker

Accessibility considerations[p154]:

For authors.
For implementers.

Sanitization[p154]:

Default[p1249] with attributes reversed[p248], start[p248], type[p248].

DOM interface[p154]:

IDL

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

  [CEReactions, Reflect] attribute boolean reversed;
  [CEReactions, Reflect, ReflectDefault=1] attribute long start;
  [CEReactions, Reflect] attribute DOMString type;

  // also has obsolete members
};
```

The ol[p247] element represents[p148] 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:
    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.
3. Return 1.

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.

Keyword	State	Description	Examples for values 1-3 and 3999-4001							
1 (U+0031)	**decimal**	Decimal numbers	1.	2.	3.	...	3999.	4000.	4001.	...
a (U+0061)	**lower-alpha**	Lowercase latin alphabet	a.	b.	c.	...	ewu.	ewv.	eww.	...
A (U+0041)	**upper-alpha**	Uppercase latin alphabet	A.	B.	C.	...	EWU.	EWV.	EWW.	...
i (U+0069)	**lower-roman**	Lowercase roman numerals	i.	ii.	iii.	...	mmmcmxcix.	$\overline{\text{iv}}$.	$\overline{\text{iv}}$i.	...
I (U+0049)	**upper-roman**	Uppercase roman numerals	I.	II.	III.	...	MMMCMXCIX.	$\overline{\text{IV}}$.	$\overline{\text{IV}}$I.	...

User agents should render the items of the list in a manner consistent with the state of the type attribute of the ol element. Numbers less than or equal to zero should always use the decimal system regardless of the type attribute.

Note

For CSS user agents, a mapping for this attribute to the 'list-style-type' CSS property is given in the Rendering section (the mapping is straightforward: the states above have the same names as their corresponding CSS values).

Note

It is possible to redefine the default CSS list styles used to implement this attribute in CSS user agents; doing so will affect how list items are rendered.

Note

Due to [ReflectDefault] 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.

Example

The following markup shows a list where the order matters, and where the ol element is therefore appropriate. Compare this list to the equivalent list in the ul section to see an example of the same items using the ul element.

```
<p>I have lived in the following countries (given in the order of when
I first lived there):</p>
<ol>
 <li>Switzerland
 <li>United Kingdom
 <li>United States
 <li>Norway
```

```
</ol>
```

Note how changing the order of the list changes the meaning of the document. In the following example, changing the relative order of the first two items has changed the birthplace of the author:

```
<p>I have lived in the following countries (given in the order of when
I first lived there):</p>
<ol>
 <li>United Kingdom
 <li>Switzerland
 <li>United States
 <li>Norway
</ol>
```

4.4.6 The ul element §[p249]

Categories[p154]:

Flow content[p157].
If the element's children include at least one li[p251] element: Palpable content[p158].

Contexts in which this element can be used[p154]:

Where flow content[p157] is expected.

Content model[p154]:

Zero or more li[p251] and script-supporting[p159] elements.

Tag omission in text/html[p154]:

Neither tag is omissible.

Content attributes[p154]:

Global attributes[p162]

Accessibility considerations[p154]:

For authors.
For implementers.

Sanitization[p154]:

Default[p1249].

DOM interface[p154]:

IDL
```
[Exposed=Window]
interface HTMLUListElement : HTMLElement {
  [HTMLConstructor] constructor();

  // also has obsolete members
};
```

The ul[p249] element represents[p148] 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[p251] element child nodes of the ul[p249] element.

Example

The following markup shows a list where the order does not matter, and where the ul[p249] element is therefore appropriate. Compare this list to the equivalent list in the ol[p247] section to see an example of the same items using the ol[p247] element.

```
<p>I have lived in the following countries:</p>
```

```
<ul>
 <li>Norway
 <li>Switzerland
 <li>United Kingdom
 <li>United States
</ul>
```

Note that changing the order of the list does not change the meaning of the document. The items in the snippet above are given in alphabetical order, but in the snippet below they are given in order of the size of their current account balance in 2007, without changing the meaning of the document whatsoever:

```
<p>I have lived in the following countries:</p>
<ul>
 <li>Switzerland
 <li>Norway
 <li>United Kingdom
 <li>United States
</ul>
```

4.4.7 The menu element §[p250]

Categories[p154]:

Flow content[p157].
If the element's children include at least one li[p251] element: Palpable content[p158].

Contexts in which this element can be used[p154]:

Where flow content[p157] is expected.

Content model[p154]:

Zero or more li[p251] and script-supporting[p159] elements.

Tag omission in text/html[p154]:

Neither tag is omissible.

Content attributes[p154]:

Global attributes[p162]

Accessibility considerations[p154]:

For authors.
For implementers.

Sanitization[p154]:

Default[p1249].

DOM interface[p154]:

IDL
```
[Exposed=Window]
interface HTMLMenuElement : HTMLElement {
  [HTMLConstructor] constructor();

  // also has obsolete members
};
```

The menu[p250] element represents[p148] a toolbar consisting of its contents, in the form of an unordered list of items (represented by li[p251] elements), each of which represents a command that the user can perform or activate.

Note

The menu[p250] element is simply a semantic alternative to ul[p249] 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:

Global 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.

Sanitization:

Default with attributes value.

DOM interface:

IDL

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

  [CEReactions, Reflect] 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`[p247], `ul`[p249], or `menu`[p250] 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[p251] *owner*, perform the following steps:

1. Let *i* be 1.
2. If *owner* is an `ol`[p247] element, let *numbering* be *owner*'s starting value[p248]. Otherwise, let *numbering* be 1.
3. *Loop*: If *i* is greater than the number of list items that *owner* owns[p251], then return; all of *owner*'s owned list items[p251] have been assigned ordinal values[p252].
4. Let *item* be the *i*th of *owner*'s owned list items[p251], in tree order.
5. If *item* is an `li`[p251] element that has a `value`[p251] attribute:
    1. Let *parsed* be the result of parsing the value of the attribute as an integer[p80].
    2. If *parsed* is not an error, then set *numbering* to *parsed*.
6. The ordinal value[p252] of *item* is *numbering*.
7. If *owner* is an `ol`[p247] element, and *owner* has a `reversed`[p248] attribute, decrement *numbering* by 1; otherwise, increment *numbering* by 1.
8. Increment *i* by 1.
9. Go to the step labeled *loop*.

Example

The element's `value`[p251] IDL attribute does not directly correspond to its ordinal value[p252]; it simply reflects[p108] the content attribute. For example, given this list:

```
<ol>
 <li>Item 1
 <li value="3">Item 3
 <li>Item 4
</ol>
```

The ordinal values[p252] are 1, 3, and 4, whereas the `value`[p251] 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`[p259] element and its `figcaption`[p262] element.

```
<figure>
 <figcaption>The top 10 movies of all time</figcaption>
 <ol>
  <li value="10"><cite>Josie and the Pussycats</cite>, 2001</li>
  <li value="9"><cite lang="sh">Црна мачка, бели мачор</cite>, 1998</li>
  <li value="8"><cite>A Bug's Life</cite>, 1998</li>
  <li value="7"><cite>Toy Story</cite>, 1995</li>
  <li value="6"><cite>Monsters, Inc</cite>, 2001</li>
  <li value="5"><cite>Cars</cite>, 2006</li>
  <li value="4"><cite>Toy Story 2</cite>, 1999</li>
  <li value="3"><cite>Finding Nemo</cite>, 2003</li>
  <li value="2"><cite>The Incredibles</cite>, 2004</li>
```

```
  <li value="1"><cite>Ratatouille</cite>, 2007</li>
 </ol>
</figure>
```

The markup could also be written as follows, using the reversed[p248] attribute on the ol[p247] element:

```
<figure>
 <figcaption>The top 10 movies of all time</figcaption>
 <ol reversed>
  <li><cite>Josie and the Pussycats</cite>, 2001</li>
  <li><cite lang="sh">Црна мачка, бели мачор</cite>, 1998</li>
  <li><cite>A Bug's Life</cite>, 1998</li>
  <li><cite>Toy Story</cite>, 1995</li>
  <li><cite>Monsters, Inc</cite>, 2001</li>
  <li><cite>Cars</cite>, 2006</li>
  <li><cite>Toy Story 2</cite>, 1999</li>
  <li><cite>Finding Nemo</cite>, 2003</li>
  <li><cite>The Incredibles</cite>, 2004</li>
  <li><cite>Ratatouille</cite>, 2007</li>
 </ol>
</figure>
```

Note

While it is conforming to include heading elements (e.g. h1[p224]) inside li[p251] 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 §[p253]

Categories[p154]:

Flow content[p157].

If the element's children include at least one name-value group: Palpable content[p158].

Contexts in which this element can be used[p154]:

Where flow content[p157] is expected.

Content model[p154]:

Either: Zero or more groups each consisting of one or more dt[p257] elements followed by one or more dd[p258] elements, optionally intermixed with script-supporting elements[p159].

Or: One or more div[p266] elements, optionally intermixed with script-supporting elements[p159].

Tag omission in text/html[p154]:

Neither tag is omissible.

Content attributes[p154]:

Global attributes[p162]

Accessibility considerations[p154]:

For authors.

For implementers.

Sanitization[p154]:

Default[p1249].

DOM interface[p154]:

IDL

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

```
  // also has obsolete members
};
```

The dl[p253] element represents[p148] 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[p257] elements, possibly as children of a div[p266] element child) followed by one or more values (dd[p258] elements, possibly as children of a div[p266] element child), ignoring any nodes other than dt[p257] and dd[p258] element children, and dt[p257] and dd[p258] elements that are children of div[p266] element children. Within a single dl[p253] element, there should not be more than one dt[p257] 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[p258] 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[p826] attributes, or other global attributes[p162] that apply to whole groups, or just for styling purposes, each group in a dl[p253] element can be wrapped in a div[p266] element. This does not change the semantics of the dl[p253] element.

The name-value groups of a dl[p253] element *dl* are determined using the following algorithm. A name-value group has a name (a list of dt[p257] elements, initially empty) and a value (a list of dd[p258] 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[p266] element:
        1. Let *grandchild* be *child*'s first child.
        2. While *grandchild* is not null:
            1. Process dt or dd[p254] for *grandchild*.
            2. Set *grandchild* to *grandchild*'s next sibling.
    2. Otherwise, process dt or dd[p254] for *child*.
    3. Set *child* to *child*'s next sibling.
7. If *current* is not empty, then append *current* to *groups*.
8. 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[p257] element:
    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[p258] 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[p258] elements in the place of dt[p257] 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").

```
<dl>
 <dt> Authors
 <dd> John
 <dd> Luke
 <dt> Editor
 <dd> Frank
</dl>
```

Example

In the following example, one definition is linked to two terms.

```
<dl>
 <dt lang="en-US"> <dfn>color</dfn> </dt>
 <dt lang="en-GB"> <dfn>colour</dfn> </dt>
 <dd> A sensation which (in humans) derives from the ability of
 the fine structure of the eye to distinguish three differently
 filtered analyses of a view. </dd>
</dl>
```

Example

The following example illustrates the use of the dl[p253] 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[p266] element around the groups of dt[p257] and dd[p258] element, to aid with styling.

```
<dl>
 <div>
  <dt> Last modified time </dt>
  <dd> 2004-12-23T23:33Z </dd>
 </div>
 <div>
  <dt> Recommended update interval </dt>
  <dd> 60s </dd>
 </div>
 <div>
  <dt> Authors </dt>
  <dt> Editors </dt>
  <dd> Robert Rothman </dd>
  <dd> Daniel Jackson </dd>
 </div>
</dl>
```

Example

The following example shows the dl[p253] 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).

```
<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[p253] element being used as a glossary. Note the use of dfn[p278] to indicate the word being defined.

```
<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[p826] attributes in a dl[p253] element, together with the div[p266] element, to annotate the ice cream desserts at a French restaurant.

```
<dl>
 <div itemscope itemtype="http://schema.org/Product">
  <dt itemprop="name">Café ou Chocolat Liégeois
  <dd itemprop="offers" itemscope itemtype="http://schema.org/Offer">
   <span itemprop="price">3.50</span>
   <data itemprop="priceCurrency" value="EUR">€</data>
  <dd itemprop="description">
   2 boules Café ou Chocolat, 1 boule Vanille, sauce café ou chocolat, chantilly
 </div>

 <div itemscope itemtype="http://schema.org/Product">
  <dt itemprop="name">Américaine
  <dd itemprop="offers" itemscope itemtype="http://schema.org/Offer">
   <span itemprop="price">3.50</span>
   <data itemprop="priceCurrency" value="EUR">€</data>
  <dd itemprop="description">
   1 boule Crème brûlée, 1 boule Vanille, 1 boule Caramel, chantilly
 </div>
</dl>
```

Without the div[p266] element the markup would need to use the itemref[p832] attribute to link the data in the dd[p258] elements with the item, as follows.

```
<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 id="1-description" itemprop="description">
  2 boules Café ou Chocolat, 1 boule Vanille, sauce café ou chocolat, chantilly
```

```
 <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 id="2-description" itemprop="description">
  1 boule Crème brûlée, 1 boule Vanille, 1 boule Caramel, chantilly
</dl>
```

Note

The dl[p253] element is inappropriate for marking up dialogue. See some examples of how to mark up dialogue[p815].

4.4.10 The dt element §[p257]

Categories[p154]:

None.

Contexts in which this element can be used[p154]:

Before dd[p258] or dt[p257] elements inside dl[p253] elements.
Before dd[p258] or dt[p257] elements inside div[p266] elements that are children of a dl[p253] element.

Content model[p154]:

Flow content[p157], but with no header[p226], footer[p227], sectioning content[p157], or heading content[p157] descendants.

Tag omission in text/html[p154]:

A dt[p257] element's end tag[p1359] can be omitted if the dt[p257] element is immediately followed by another dt[p257] element or a dd[p258] element.

Content attributes[p154]:

Global attributes[p162]

Accessibility considerations[p154]:

For authors.
For implementers.

Sanitization[p154]:

Default[p1249].

DOM interface[p154]:

Uses HTMLElement[p149].

The dt[p257] element represents[p148] the term, or name, part of a term-description group in a description list (dl[p253] element).

Note

The dt[p257] element itself, when used in a dl[p253] element, does not indicate that its contents are a term being defined, but this can be indicated using the dfn[p278] element.

Example

This example shows a list of frequently asked questions (a FAQ) marked up using the dt[p257] element for questions and the dd[p258] element for answers.

```
<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 dd element §[p258]

Categories[p154]:

None.

Contexts in which this element can be used[p154]:

After dt[p257] or dd[p258] elements inside dl[p253] elements.
After dt[p257] or dd[p258] elements inside div[p266] elements that are children of a dl[p253] element.

Content model[p154]:

Flow content[p157].

Tag omission in text/html[p154]:

A dd[p258] element's end tag[p1359] can be omitted if the dd[p258] element is immediately followed by another dd[p258] element or a dt[p257] element, or if there is no more content in the parent element.

Content attributes[p154]:

Global attributes[p162]

Accessibility considerations[p154]:

For authors.
For implementers.

Sanitization[p154]:

Default[p1249].

DOM interface[p154]:

Uses HTMLElement[p149].

The dd[p258] element represents[p148] the description, definition, or value, part of a term-description group in a description list (dl[p253] element).

Example

A dl[p253] can be used to define a vocabulary list, like in a dictionary. In the following example, each entry, given by a dt[p257] with a dfn[p278], has several dd[p258]s, showing the various parts of the definition.

```
<dl>
 <dt><dfn>happiness</dfn></dt>
 <dd class="pronunciation">/'hæpinəs/</dd>
 <dd class="part-of-speech"><i><abbr>n.</abbr></i></dd>
 <dd>The state of being happy.</dd>
 <dd>Good fortune; success. <q>Oh <b>happiness</b>! It worked!</q></dd>
 <dt><dfn>rejoice</dfn></dt>
 <dd class="pronunciation">/rɪ'dʒɔɪs/</dd>
 <dd><i class="part-of-speech"><abbr>v.intr.</abbr></i> To be delighted oneself.</dd>
 <dd><i class="part-of-speech"><abbr>v.tr.</abbr></i> To cause one to be delighted.</dd>
</dl>
```

4.4.12 The figure element §[p259]

Categories[p154]:

Flow content[p157].
Palpable content[p158].

Contexts in which this element can be used[p154]:

Where flow content[p157] is expected.

Content model[p154]:

Either: one figcaption[p262] element followed by flow content[p157].
Or: flow content[p157] followed by one figcaption[p262] element.
Or: flow content[p157].

Tag omission in text/html[p154]:

Neither tag is omissible.

Content attributes[p154]:

Global attributes[p162]

Accessibility considerations[p154]:

For authors.
For implementers.

Sanitization[p154]:

Default[p1249].

DOM interface[p154]:

Uses HTMLElement[p149].

The figure[p259] element represents[p148] some flow content[p157], optionally with a caption, that is self-contained (like a complete sentence) and is typically referenced[p148] 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[p259], but an entire sentence made of images would be fitting.

The element can thus be used to annotate illustrations, diagrams, photos, code listings, etc.

¶[p259]

Note

When a figure[p259] 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[p259] element is referenced[p148] 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[p262] element child of the element, if any, represents the caption of the figure[p259] element's contents. If there is no child figcaption[p262] element, then there is no caption.

A figure[p259] 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[p259] and figcaption[p262] 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[p222] element should be used (and can itself wrap a figure[p259]). For example, a pull quote that repeats content from an article[p214] would be more appropriate in an aside[p222] than in a figure[p259], 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[p259] element to mark up a code listing.

```
<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> data);
 undefined initSelfDestruct();
}</code></pre>
</figure>
<p>The API is designed to use UTF-8.</p>
```

Example

Here we see a figure[p259] element to mark up a photo that is the main content of the page (as in a gallery).

```
<!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>
<nav><a href="19414.html">Prev</a> — <a href="19416.html">Next</a></nav>
```

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[p259] would be inappropriate.

```
<h2>Malinko's comics</h2>

<p>This case centered on some sort of "intellectual property"
infringement related to a comic (see Exhibit A). The suit started
after a trailer ending with these words:

<blockquote>
 <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.

<figure>
 <img src="ex-a.png" alt="Two squiggles on a dirty piece of paper.">
 <figcaption>Exhibit A. The alleged <cite>rough copy</cite> comic.</figcaption>
</figure>

<figure>
 <video src="ex-b.mov"></video>
 <figcaption>Exhibit B. The <cite>Rough Copy</cite> trailer.</figcaption>
</figure>

<p>The case was resolved out of court.
```

Example

Here, a part of a poem is marked up using figure[p259].

```
<figure>
 <p>'Twas brillig, and the slithy toves<br>
 Did gyre and gimble in the wabe;<br>
 All mimsy were the borogoves,<br>
 And the mome raths outgrabe.</p>
 <figcaption><cite>Jabberwocky</cite> (first verse). Lewis Carroll, 1832-98</figcaption>
</figure>
```

Example

In this example, which could be part of a much larger work discussing a castle, nested figure[p259] elements are used to provide both a group caption and individual captions for each figure in the group:

```
<figure>
 <figcaption>The castle through the ages: 1423, 1858, and 1999 respectively.</figcaption>
 <figure>
  <figcaption>Etching. Anonymous, ca. 1423.</figcaption>
  <img src="castle1423.jpeg" alt="The castle has one tower, and a tall wall around it.">
 </figure>
 <figure>
  <figcaption>Oil-based paint on canvas. Maria Towle, 1858.</figcaption>
  <img src="castle1858.jpeg" alt="The castle now has two towers and two walls.">
 </figure>
 <figure>
  <figcaption>Film photograph. Peter Jankle, 1999.</figcaption>
  <img src="castle1999.jpeg" alt="The castle lies in ruins, the original tower all that remains in
one piece.">
 </figure>
</figure>
```

Example

The previous example could also be more succinctly written as follows (using title[p164] attributes in place of the nested figure[p259]/figcaption[p262] pairs):

```
<figure>
 <img src="castle1423.jpeg" title="Etching. Anonymous, ca. 1423."
      alt="The castle has one tower, and a tall wall around it.">
 <img src="castle1858.jpeg" title="Oil-based paint on canvas. Maria Towle, 1858."
      alt="The castle now has two towers and two walls.">
 <img src="castle1999.jpeg" title="Film photograph. Peter Jankle, 1999."
      alt="The castle lies in ruins, the original tower all that remains in one piece.">
 <figcaption>The castle through the ages: 1423, 1858, and 1999 respectively.</figcaption>
</figure>
```

Example

The figure is sometimes referenced[p148] only implicitly from the content:

```
<article>
 <h1>Fiscal negotiations stumble in Congress as deadline nears</h1>
 <figure>
  <img src="obama-reid.jpeg" alt="Obama and Reid sit together smiling in the Oval Office.">
  <figcaption>Barack Obama and Harry Reid. White House press photograph.</figcaption>
 </figure>
 <p>Negotiations in Congress to end the fiscal impasse sputtered on Tuesday, leaving both chambers
 grasping for a way to reopen the government and raise the country's borrowing authority with a
 Thursday deadline drawing near.</p>
```

```
  ...
</article>
```

4.4.13 The figcaption element §[p26]

Categories[p154]:

None.

Contexts in which this element can be used[p154]:

As the first or last child of a figure[p259] element.

Content model[p154]:

Flow content[p157].

Tag omission in text/html[p154]:

Neither tag is omissible.

Content attributes[p154]:

Global attributes[p162]

Accessibility considerations[p154]:

For authors.
For implementers.

Sanitization[p154]:

Default[p1249].

DOM interface[p154]:

Uses HTMLElement[p149].

The figcaption[p262] element represents[p148] a caption or legend for the rest of the contents of the figcaption[p262] element's parent figure[p259] element, if any.

Example

The element can contain additional information about the source:

```
<figcaption>
 <p>A duck.</p>
 <p><small>Photograph courtesy of ☆ News.</small></p>
</figcaption>
```

```
<figcaption>
 <p>Average rent for 3-room apartments, excluding non-profit apartments</p>
 <p>Zürich's Statistics Office — <time datetime=2017-11-14>14 November 2017</time></p>
</figcaption>
```

4.4.14 The main element §[p26]

Categories[p154]:

Flow content[p157].
Palpable content[p158].

Contexts in which this element can be used[p154]:

Where flow content[p157] is expected, but only if it is a hierarchically correct main element[p263].

Content model[p154]:

Flow content[p157].

Tag omission in text/html[p154]:

Neither tag is omissible.

Content attributes[p154]:

Global attributes[p162]

Accessibility considerations[p154]:

For authors.
For implementers.

Sanitization[p154]:

Default[p1249].

DOM interface[p154]:

Uses `HTMLElement`[p149].

The `main`[p262] element represents[p148] the dominant contents of the document.

A document must not have more than one `main`[p262] element that does not have the `hidden`[p862] attribute specified.

A **hierarchically correct `main` element** is one whose ancestor elements are limited to `html`[p179], `body`[p212], `div`[p266], `form`[p532] without an accessible name, and autonomous custom elements[p796]. Each `main`[p262] element must be a hierarchically correct `main` element[p263].

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`[p262] element is used.

```
<!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.
</aside>
<main>
 <h2>Character creation</h2>
 <p>Attributes (magic, strength, agility) are purchased at the cost of one point per level.</p>
 <h2>Rolls</h2>
 <p>Each encounter, roll the dice for all your skills. If you roll more than the opponent, you
win.</p>
</main>
<footer>
 <p>Copyright © 2013
</footer>
</html>
```

In the following example, multiple main[p262] elements are used and script is used to make navigation work without a server roundtrip and to set the hidden[p862] attribute on those that are not current:

```
<!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</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 search element §[p264]

Categories[p154]:

Flow content[p157].
Palpable content[p158].

Contexts in which this element can be used[p154]:

Where flow content[p157] is expected.

Content model[p154]:

Flow content[p157].

Tag omission in text/html[p154]:

Neither tag is omissible.

Content attributes[p154]:

Global attributes[p162]

Accessibility considerations[p154]:

For authors.
For implementers.

Sanitization[p154]:

Default[p1249].

DOM interface[p154]:

Uses HTMLElement[p149].

The search[p264] element represents[p148] a part of a document or application that contains a set of form controls or other content related to performing a search or filtering operation. This could be a search of the web site or application; a way of searching or filtering search results on the current web page; or a global or Internet-wide search function.

Note

It's not appropriate to use the search[p264] element just for presenting search results, though suggestions and links as part of "quick search" results can be included as part of a search feature. Rather, a returned web page of search results would instead be expected to be presented as part of the main content of that web page.

Example

In the following example, the author is including a search form within the header[p226] of the web page:

```
<header>
  <h1><a href="/">My fancy blog</a></h1>
  ...
  <search>
    <form action="search.php">
      <label for="query">Find an article</label>
      <input id="query" name="q" type="search">
      <button type="submit">Go!</button>
    </form>
  </search>
</header>
```

Example

In this example, the author has implemented their web application's search functionality entirely with JavaScript. There is no use of the form[p532] element to perform server-side submission, but the containing search[p264] element semantically identifies the purpose of the descendant content as representing search capabilities.

```
<search>
  <label>
    Find and filter your query
    <input type="search" id="query">
  </label>
  <label>
    <input type="checkbox" id="exact-only">
    Exact matches only
  </label>

  <section>
    <h3>Results found:</h3>
    <ul id="results">
      <li>
        <p><a href="services/consulting">Consulting services</a></p>
        <p>
          Find out how can we help you improve your business with our integrated consultants, Bob
and Bob.
        </p>
      </li>
      ...
    </ul>
    <!--
      when a query returns or filters out all results
      render the no results message here
    -->
    <output id="no-results"></output>
  </section>
</search>
```

Example

In the following example, the page has two search features. The first is located in the web page's header[p226] and serves as a global mechanism to search the web site's content. Its purpose is indicated by its specified title[p181] attribute. The second is included as

part of the main content of the page, as it represents a mechanism to search and filter the content of the current page. It contains a heading to indicate its purpose.

```
<body>
  <header>
    ...
    <search title="Website">
      ...
    </search>
  </header>
  <main>
    <h1>Hotels near your location</h1>
     <search>
       <h2>Filter results</h2>
       ...
     </search>
     <article>
      <!-- search result content -->
    </article>
  </main>
</body>
```

4.4.16 The div element §[p266]

Categories[p154]:

Flow content[p157].
Palpable content[p158].

Contexts in which this element can be used[p154]:

Where flow content[p157] is expected.
As a child of a dl[p253] element.
As a descendant of an option[p601] element, an optgroup[p600] element, or a select[p590] element.

Content model[p154]:

If the element is a child of a dl[p253] element: One or more dt[p257] elements followed by one or more dd[p258] elements, optionally intermixed with script-supporting elements[p159].
Otherwise, if the element is a descendant of an option[p601] element, an optgroup[p600] element, or a select[p590] element: transparent[p159].
Otherwise: flow content[p157].

Tag omission in text/html[p154]:

Neither tag is omissible.

Content attributes[p154]:

Global attributes[p162]

Accessibility considerations[p154]:

For authors.
For implementers.

Sanitization[p154]:

Default[p1249].

DOM interface[p154]:

IDL

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

  // also has obsolete members
```

```
};
```

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:

```
<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>
  <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>
 </div>
 <p>I should say "sidewalk" and "apartment" and "color"!</p>
</article>
```

4.5 Text-level semantics §

4.5.1 The a element §

Categories:

Flow content.
Phrasing content.
If the element has an href attribute: Interactive content.
Palpable content.

Contexts in which this element can be used:

Where phrasing content is expected.

Content model:

Transparent, but there must be no interactive content descendant, a element descendant, or descendant with the tabindex attribute specified.

Tag omission in text/html:

Neither tag is omissible.

Content attributes:

Global attributes

href[p314] — Address of the hyperlink[p313]
target[p314] — Navigable[p1035] for hyperlink[p313] navigation[p1063]
download[p314] — Whether to download the resource instead of navigating to it, and its filename if so
ping[p314] — URLs to ping
rel[p314] — Relationship between the location in the document containing the hyperlink[p313] and the destination resource
hreflang[p316] — Language of the linked resource
type[p316] — Hint for the type of the referenced resource
referrerpolicy[p314] — Referrer policy for fetches initiated by the element

Accessibility considerations[p154]:

If the element has an href[p314] attribute: for authors; for implementers.
Otherwise: for authors; for implementers.

Sanitization[p154]:

Default[p1249] with attributes href[p314], hreflang[p316], and type[p316], and navigating URL attributes[p1251] href[p314].

DOM interface[p154]:

IDL

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

  [CEReactions, Reflect] attribute DOMString download;
  [CEReactions, Reflect] attribute USVString ping;
  [CEReactions, Reflect] attribute DOMString rel;
  [SameObject, PutForwards=value, Reflect="rel"] readonly attribute DOMTokenList relList;

  [CEReactions] attribute DOMString text;

  [CEReactions] attribute DOMString referrerPolicy;

  // also has obsolete members
};
HTMLAnchorElement includes HyperlinkElementUtils;
HTMLAnchorElement includes HTMLHyperlinkElementUtils;
```

If the a[p267] element has an href[p314] attribute, then it represents[p148] a hyperlink[p313] (a hypertext anchor) labeled by its contents.

If the a[p267] element has no href[p314] attribute, then the element represents[p148] 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[p314], download[p314], ping[p314], rel[p314], hreflang[p316], type[p316], and referrerpolicy[p314] attributes must be omitted if the href[p314] attribute is not present.

If the itemprop[p833] attribute is specified on an a[p267] element, then the href[p314] 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 a[p267] element:

```
<nav>
 <ul>
  <li> <a href="/">Home</a> </li>
  <li> <a href="/news">News</a> </li>
  <li> <a>Examples</a> </li>
  <li> <a href="/legal">Legal</a> </li>
 </ul>
</nav>
```

The href[p314], target[p314], download[p314], ping[p314], and referrerpolicy[p314] attributes affect what happens when users follow hyperlinks[p321] or download hyperlinks[p322] created using the a[p267] element. The rel[p314], hreflang[p316], and type[p316] attributes may be used to indicate to the user the likely nature of the target resource before the user follows the link.

For web developers (non-normative)

***a*.text**[p269]

Same as textContent.

The IDL attribute **referrerPolicy** must reflect[p108] the referrerpolicy[p314] content attribute, limited to only known values[p109].

The **text** attribute's getter must return this element's descendant text content.

The text[p269] attribute's setter must string replace all with the given value within this element.

Example

The a[p267] 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:

```
<aside class="advertising">
 <h1>Advertising</h1>
 <a href="https://ad.example.com/?adid=1929&amp;pubid=1422">
  <section>
   <h1>Mellblomatic 9000!</h1>
   <p>Turn all your widgets into mellbloms!</p>
   <p>Only $9.99 plus shipping and handling.</p>
  </section>
 </a>
 <a href="https://ad.example.com/?adid=375&amp;pubid=1422">
  <section>
   <h1>The Mellblom Browser</h1>
   <p>Web browsing at the speed of light.</p>
   <p>No other browser goes faster!</p>
  </section>
 </a>
</aside>
```

Example

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:

```
<table>
 <tr>
  <th>Position
  <th>Team
  <th>Location
 <tr>
  <td><a href="/jobs/manager">Manager</a>
  <td>Remotees
  <td>Remote
 <tr>
  <td><a href="/jobs/director">Director</a>
  <td>Remotees
  <td>Remote
 <tr>
  <td><a href="/jobs/astronaut">Astronaut</a>
  <td>Architecture
  <td>Remote
</table>
<script>
document.querySelector("table").onclick = ({ target }) => {
  if (target.parentElement.localName === "tr") {
    const link = target.parentElement.querySelector("a");
    if (link) {
      link.click();
```

```
    }
  }
}
</script>
```

4.5.2 The em element §[p27]

Categories[p154]:

Flow content[p157].
Phrasing content[p157].
Palpable content[p158].

Contexts in which this element can be used[p154]:

Where phrasing content[p157] is expected.

Content model[p154]:

Phrasing content[p157].

Tag omission in text/html[p154]:

Neither tag is omissible.

Content attributes[p154]:

Global attributes[p162]

Accessibility considerations[p154]:

For authors.
For implementers.

Sanitization[p154]:

Default[p1249].

DOM interface[p154]:

Uses HTMLElement[p149].

The em[p270] element represents[p148] stress emphasis of its contents.

The level of stress that a particular piece of content has is given by its number of ancestor em[p270] 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 <em>cute</em> animals!</em></p>
```

Note

The em[p270] 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[p302] element is more appropriate.

The em[p270] element also isn't intended to convey importance; for that purpose, the strong[p271] element is more appropriate.

✔ MDN

4.5.3 The strong element §[p271]

Categories[p154]:

Flow content[p157].
Phrasing content[p157].
Palpable content[p158].

Contexts in which this element can be used[p154]:

Where phrasing content[p157] is expected.

Content model[p154]:

Phrasing content[p157].

Tag omission in text/html[p154]:

Neither tag is omissible.

Content attributes[p154]:

Global attributes[p162]

Accessibility considerations[p154]:

For authors.
For implementers.

Sanitization[p154]:

Default[p1249].

DOM interface[p154]:

Uses HTMLElement[p149].

The strong[p271] element represents[p148] strong importance, seriousness, or urgency for its contents.

Importance: the strong[p271] 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[p225] element is appropriate.)

Example

For example, the first word of the previous paragraph is marked up with strong[p271] to distinguish it from the more detailed text in the rest of the paragraph.

Seriousness: the strong[p271] element can be used to mark up a warning or caution notice.

Urgency: the strong[p271] 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[p271] elements; each strong[p271] element increases the importance of its contents.

Changing the importance of a piece of text with the strong[p271] 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[p271]:

```
<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[p271], to distinguish it from boilerplate text (before) and the description (after):

```
<figcaption>Figure 1. <strong>Ant colony dynamics</strong>. The ants in this colony are
affected by the heat source (upper left) and the food source (lower right).</figcaption>
```

In this example, the heading is really "Flowers, Bees, and Honey", but the author has added a light-hearted addition to the heading. The strong[p271] element is thus used to mark up the first part to distinguish it from the latter part.

```
<h1><strong>Flowers, Bees, and Honey</strong> and other things I don't understand</h1>
```

Example

Here is an example of a warning notice in a game, with the various parts marked up according to how important they are:

```
<p><strong>Warning.</strong> This dungeon is dangerous.
<strong>Avoid the ducks.</strong> Take any gold you find.
<strong><strong>Do not take any of the diamonds</strong>,
they are explosive and <strong>will destroy anything within
ten meters.</strong></strong> You have been warned.</p>
```

Example

In this example, the strong[p271] element is used to denote the part of the text that the user is intended to read first.

```
<p>Welcome to Remy, the reminder system.</p>
<p>Your tasks for today:</p>
<ul>
 <li><p><strong>Turn off the oven.</strong></p></li>
 <li><p>Put out the trash.</p></li>
 <li><p>Do the laundry.</p></li>
</ul>
```

4.5.4 The small element §[p272]

Categories[p154]:

Flow content[p157].
Phrasing content[p157].

Palpable content[p158].

Contexts in which this element can be used[p154]:

Where phrasing content[p157] is expected.

Content model[p154]:

Phrasing content[p157].

Tag omission in text/html[p154]:

Neither tag is omissible.

Content attributes[p154]:

Global attributes[p162]

Accessibility considerations[p154]:

For authors.
For implementers.

Sanitization[p154]:

Default[p1249].

DOM interface[p154]:

Uses `HTMLElement`[p149].

The `small`[p272] element represents[p148] 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`[p272] element does not "de-emphasize" or lower the importance of text emphasized by the `em`[p270] element or marked as important with the `strong`[p271] element. To mark text as not emphasized or important, simply do not mark it up with the `em`[p270] or `strong`[p271] elements respectively.

The `small`[p272] 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`[p272] element: in such a case, the text is not a side comment, it is the main content of the page.

The `small`[p272] element must not be used for subheadings; for that purpose, use the `hgroup`[p225] element.

Example

In this example, the `small`[p272] element is used to indicate that value-added tax is not included in a price of a hotel room:

Example

```
<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`[p272] element is used for a side comment in an article.

```
<p>Example Corp today announced record profits for the
second quarter <small>(Full Disclosure: Foo News is a subsidiary of
Example Corp)</small>, leading to speculation about a third quarter
merger with Demo Group.</p>
```

This is distinct from a sidebar, which might be multiple paragraphs long and is removed from the main flow of text. In the following example, we see a sidebar from the same article. This sidebar also has small print, indicating the source of the information in the sidebar.

```
<aside>
 <h1>Example Corp</h1>
 <p>This company mostly creates small software and Web
 sites.</p>
 <p>The Example Corp company mission is "To provide entertainment
 and news on a sample basis".</p>
 <p><small>Information obtained from <a
 href="https://example.com/about.html">example.com</a> home
 page.</small></p>
</aside>
```

Example

In this last example, the small[p272] element is marked as being *important* small print.

```
<p><strong><small>Continued use of this service will result in a kiss.</small></strong></p>
```

4.5.5 The s element §[p274]

Categories[p154]:

Flow content[p157].
Phrasing content[p157].
Palpable content[p158].

Contexts in which this element can be used[p154]:

Where phrasing content[p157] is expected.

Content model[p154]:

Phrasing content[p157].

Tag omission in text/html[p154]:

Neither tag is omissible.

Content attributes[p154]:

Global attributes[p162]

Accessibility considerations[p154]:

For authors.
For implementers.

Sanitization[p154]:

Default[p1249].

DOM interface[p154]:

Uses HTMLElement[p149].

The s[p274] element represents[p148] contents that are no longer accurate or no longer relevant.

Note

The s[p274] element is not appropriate when indicating document edits; to mark a span of text as having been removed from a document, use the del[p351] 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.

```
<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 §[p275]

Categories[p154]:

Flow content[p157].
Phrasing content[p157].
Palpable content[p158].

Contexts in which this element can be used[p154]:

Where phrasing content[p157] is expected.

Content model[p154]:

Phrasing content[p157].

Tag omission in text/html[p154]:

Neither tag is omissible.

Content attributes[p154]:

Global attributes[p162]

Accessibility considerations[p154]:

For authors.
For implementers.

Sanitization[p154]:

Default[p1249].

DOM interface[p154]:

Uses HTMLElement[p149].

The cite[p275] element represents[p148] 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[p148] 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[p303] 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[p309] element can be used.)

¶[p275] **Example**

This next example shows a typical use of the cite[p275] element:

```
<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>
```

¶[p275] **Example**

This is correct usage:

```
<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[p275] element here is containing far more than the title of the work:

```
<!-- 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>
```

¶[p27] 6 **Example**

The cite[p275] element is a key part of any citation in a bibliography, but it is only used to mark the title:

```
<p><cite>Universal Declaration of Human Rights</cite>, United Nations,
December 1948. Adopted by General Assembly resolution 217 A (III).</p>
```

¶[p27] 6 **Note**

A citation *is not a* quote *(for which the* q[p277] *element is appropriate).*

¶[p27] 6 **Example**

This is incorrect usage, because cite[p275] is not for quotes:

```
<p><cite>This is wrong!</cite>, said Ian.</p>
```

This is also incorrect usage, because a person is not a work:

```
<p><q>This is still wrong!</q>, said <cite>Ian</cite>.</p>
```

The correct usage does not use a cite[p275] element:

```
<p><q>This is correct</q>, said Ian.</p>
```

As mentioned above, the b[p303] element might be relevant for marking names as being keywords in certain kinds of documents:

```
<p>And then <b>Ian</b> said <q>this might be right, in a
gossip column, maybe!</q>.</p>
```

¶[p27] 6 **Note**

Where the title of a work is conventionally set off with punctuation — for example, the quotation marks used around article titles in English, or the corner brackets (『』) used around book titles in Japanese — that punctuation is not part of the title, and belongs outside the cite[p275] *element.*

¶[p27] 6 **Example**

This is correct usage:

```
<p>My review of 『<cite lang="ja">阪急電車</cite>』 was titled “<cite>A cute series of short
stories</cite>”.</p>
```

4.5.7 The q element §[p277]

Categories[p154]:

Flow content[p157].
Phrasing content[p157].
Palpable content[p158].

Contexts in which this element can be used[p154]:

Where phrasing content[p157] is expected.

Content model[p154]:

Phrasing content[p157].

Tag omission in text/html[p154]:

Neither tag is omissible.

Content attributes[p154]:

Global attributes[p162]
cite[p277] — Link to the source of the quotation or more information about the edit

Accessibility considerations[p154]:

For authors.
For implementers.

Sanitization[p154]:

Default[p1249].

DOM interface[p154]:

Uses HTMLQuoteElement[p244].

The q[p277] element represents[p148] some phrasing content[p157] quoted from another source.

Quotation punctuation (such as quotation marks) that is quoting the contents of the element must not appear immediately before, after, or inside q[p277] elements; they will be inserted into the rendering by the user agent.

Content inside a q[p277] element must be quoted from another source, whose address, if it has one, may be cited in the **cite** attribute. The source may be fictional, as when quoting characters in a novel or screenplay.

If the cite[p277] attribute is present, it must be a valid URL potentially surrounded by spaces[p100]. To obtain the corresponding citation link, the value of the attribute must be parsed[p101] relative to the element's node document. User agents may allow users to follow such citation links, but they are primarily intended for private use (e.g., by server-side scripts collecting statistics about a site's use of quotations), not for readers.

The q[p277] element must not be used in place of quotation marks that do not represent quotes; for example, it is inappropriate to use the q[p277] element for marking up sarcastic statements.

The use of q[p277] elements to mark up quotations is entirely optional; using explicit quotation punctuation without q[p277] elements is just as correct.

Example

Here is a simple example of the use of the q[p277] element:

```
<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 q[p277] element, and an explicit citation outside:

```
<p>The W3C page <cite>About W3C</cite> says the W3C's
mission is <q cite="https://www.w3.org/Consortium/">To lead the
World Wide Web to its full potential by developing protocols and
guidelines that ensure long-term growth for the Web</q>. I
disagree with this mission.</p>
```

Example

In the following example, the quotation itself contains a quotation:

```
<p>In <cite>Example One</cite>, he writes <q>The man
said <q>Things that are impossible just take longer</q>. I
disagreed with him</q>. Well, I disagree even more!</p>
```

Example

In the following example, quotation marks are used instead of the q[p277] element:

```
<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[p277] element in this case would be inappropriate.

```
<p>The word "ineffable" could have been used to describe the disaster
resulting from the campaign's mismanagement.</p>
```

✔ MDN

4.5.8 The dfn element §[p278]

Categories[p154]:

Flow content[p157].
Phrasing content[p157].
Palpable content[p158].

Contexts in which this element can be used[p154]:

Where phrasing content[p157] is expected.

Content model[p154]:

Phrasing content[p157], but there must be no dfn[p278] element descendants.

Tag omission in text/html[p154]:

Neither tag is omissible.

Content attributes[p154]:

Global attributes[p162]
Also, the title[p278] attribute has special semantics[p278] on this element: Full term or expansion of abbreviation

Accessibility considerations[p154]:

For authors.
For implementers.

Sanitization[p154]:

Default[p1249].

DOM interface[p154]:

Uses HTMLElement[p149].

The dfn[p278] element represents[p148] the defining instance of a term. The paragraph[p159], description list group[p253], or section[p157] that is the nearest ancestor of the dfn[p278] element must also contain the definition(s) for the term[p278] given by the dfn[p278] element.

Defining term: if the dfn[p278] 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[p279] element with a title[p279] attribute, then the exact value of *that* attribute is the term being defined. Otherwise, it is the descendant text content of the dfn[p278] element that gives the term being defined.

If the title[p278] attribute of the dfn[p278] element is present, then it must contain only the term being defined.

Note

The `title`[p164] attribute of ancestor elements does not affect `dfn`[p278] elements.

An `a`[p267] element that links to a `dfn`[p278] element represents an instance of the term defined by the `dfn`[p278] 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 <dfn><abbr title="Garage Door Opener">GDO</abbr></dfn>
is a device that allows off-world teams to open the iris.</p>
<!-- ... later in the document: -->
<p>Teal'c activated his <abbr title="Garage Door Opener">GDO</abbr>
and so Hammond ordered the iris to be opened.</p>
```

With the addition of an `a`[p267] element, the reference[p148] can be made explicit:

```
<p>The <dfn id=gdo><abbr title="Garage Door Opener">GDO</abbr></dfn>
is a device that allows off-world teams to open the iris.</p>
<!-- ... later in the document: -->
<p>Teal'c activated his <a href=#gdo><abbr title="Garage Door Opener">GDO</abbr></a>
and so Hammond ordered the iris to be opened.</p>
```

✔ MDN

4.5.9 The `abbr` element §[p279]

Categories[p154]:

Flow content[p157].
Phrasing content[p157].
Palpable content[p158].

Contexts in which this element can be used[p154]:

Where phrasing content[p157] is expected.

Content model[p154]:

Phrasing content[p157].

Tag omission in text/html[p154]:

Neither tag is omissible.

Content attributes[p154]:

Global attributes[p162]
Also, the `title`[p279] attribute has special semantics[p279] on this element: Full term or expansion of abbreviation

Accessibility considerations[p154]:

For authors.
For implementers.

Sanitization[p154]:

Default[p1249].

DOM interface[p154]:

Uses `HTMLElement`[p149].

The `abbr`[p279] element represents[p148] an abbreviation or acronym, optionally with its expansion. The **`title`** attribute may be used to provide an expansion of the abbreviation. The attribute, if specified, must contain an expansion of the abbreviation, and nothing else.

Example

The paragraph below contains an abbreviation marked up with the `abbr`[p279] element. This paragraph defines the term[p278] "Web

Hypertext Application Technology Working Group".

```
<p>The <dfn id=whatwg><abbr
title="Web Hypertext Application Technology Working Group">WHATWG</abbr></dfn>
is a loose unofficial collaboration of web browser manufacturers and
interested parties who wish to develop new technologies designed to
allow authors to write and deploy Applications over the World Wide
Web.</p>
```

An alternative way to write this would be:

```
<p>The <dfn id=whatwg>Web Hypertext Application Technology
Working Group</dfn> (<abbr
title="Web Hypertext Application Technology Working Group">WHATWG</abbr>)
is a loose unofficial collaboration of web browser manufacturers and
interested parties who wish to develop new technologies designed to
allow authors to write and deploy Applications over the World Wide
Web.</p>
```

Example

This paragraph has two abbreviations. Notice how only one is defined; the other, with no expansion associated with it, does not use the abbr[p279] 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[p279] element with a title[p164] 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[p279] element with a title[p164] 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[p279] element can be used without a title[p164] attribute.

Note

Providing an expansion in a title[p164] attribute once will not necessarily cause other abbr[p279] elements in the same document with the same contents but without a title[p164] attribute to behave as if they had the same expansion. Every abbr[p279] element is independent.

4.5.10 The ruby element §[p281]

Categories[p154]:

Flow content[p157].
Phrasing content[p157].
Palpable content[p158].

Contexts in which this element can be used[p154]:

Where phrasing content[p157] is expected.

Content model[p154]:

See prose.

Tag omission in text/html[p154]:

Neither tag is omissible.

Content attributes[p154]:

Global attributes[p162]

Accessibility considerations[p154]:

For authors.
For implementers.

Sanitization[p154]:

Default[p1249].

DOM interface[p154]:

Uses HTMLElement[p149].

The ruby[p281] 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[p281] elements consists of one or more of the following sequences:

1. One or the other of the following:
    - Phrasing content[p157], but with no ruby[p281] elements and with no ruby[p281] element descendants
    - A single ruby[p281] element that itself has no ruby[p281] element descendants

2. One or the other of the following:

    - One or more rt[p287] elements
    - An rp[p288] element followed by one or more rt[p287] elements, each of which is itself followed by an rp[p288] element

The ruby[p281] and rt[p287] elements can be used for a variety of kinds of annotations, including in particular (though by no means limited to) those described below. For more details on Japanese Ruby in particular, and how to render Ruby for Japanese, see *Requirements for Japanese Text Layout*. [JLREQ][p1583]

Note

At the time of writing, CSS does not yet provide a way to fully control the rendering of the HTML ruby[p281] 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.

Example

```
<ruby>B<rt>annotation</ruby>
```

Example

In this example, notice how each annotation corresponds to a single base character.

```
<ruby>君<rt>くん</ruby><ruby>子<rt>し</ruby>は<ruby>和<rt>わ</ruby>して<ruby>同<rt>どう</ruby>ぜず。
```

君くん子しは和わして同どうぜず。

This example can also be written as follows, using one ruby[p281] element with two segments of base text and two annotations (one for each) rather than two back-to-back ruby[p281] elements each with one base text segment and annotation (as in the markup above):

```
<ruby>君<rt>くん</rt>子<rt>し</ruby>は<ruby>和<rt>わ</ruby>して<ruby>同<rt>どう</ruby>ぜず。
```

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

```
<ruby>B<rt>annotation</rt>B<rt>annotation</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[p281] 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<ruby>方<rt>ほう</rt>角<rt>がく</rt></ruby>を<ruby>凝<rt>ぎょう</rt>視<rt>し</rt></ruby>する
```

鬼き門もんの方ほう角がくを凝ぎょう視しする

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<ruby>方<rt>ほう</rt>角<rt>がく</rt></ruby>を<ruby>凝<rt>ぎょう</rt>視<rt>し</rt></ruby>する
```

Note

For more details on Jukugo Ruby rendering, see Appendix F in the Requirements for Japanese Text Layout. *[JLREQ]*[p1583]

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

```
<ruby>BASE<rt>annotation</ruby>
```

Example

Here a compound ideographic word has its corresponding katakana given as an annotation.

```
<ruby>境界面<rt>インターフェース</ruby>
```

境界面インターフェース

Example

Here a compound ideographic word has its translation in English provided as an annotation.

```
<ruby lang="ja">編集者<rt lang="en">editor</ruby>
```

編集者editor

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

In this example, the name of a species of flowers has a phonetic reading provided using group ruby:

```
<ruby>紫陽花<rt>あじさい</ruby>
```

紫陽花あじさい

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>BASE<rt>annotation 1<rt>annotation 2</ruby>
```

Example

```
<ruby>B<rt>a<rt>a</ruby><ruby>A<rt>a<rt>a</ruby><ruby>S<rt>a<rt>a</ruby><ruby>E<rt>a<rt>a</ruby>
```

Example

In this contrived example, some symbols are given names in English and French.

```
<ruby>
 ♥ <rt> Heart <rt lang=fr> Cœur </rt>
 ♣ <rt> Shamrock <rt lang=fr> Trèfle </rt>
 ✶ <rt> Star <rt lang=fr> Étoile </rt>
</ruby>
```

In more complicated situations such as the following examples, a nested ruby[p281] element is used to give the inner annotations, and then that whole ruby[p281] is then given an annotation at the "outer" level.

Example

```
<ruby><ruby>B<rt>a</rt>A<rt>n</rt>S<rt>t</rt>E<rt>n</rt></ruby><rt>annotation</ruby>
```

Example

Here both a phonetic reading and the meaning are given in ruby annotations. The annotation on the nested ruby[p281] element gives a mono-ruby phonetic annotation for each base character, while the annotation in the rt[p287] element that is a child of the outer ruby[p281] element gives the meaning using hiragana.

```
<ruby><ruby>東<rt>とう</rt>南<rt>なん</rt></ruby><rt>たつみ</rt></ruby>の方角
```

東とう南なんたつみの方角

Example

This is the same example, but the meaning is given in English instead of Japanese:

```
<ruby><ruby>東<rt>とう</rt>南<rt>なん</rt></ruby><rt lang=en>Southeast</rt></ruby>の方角
```

東とう南なんSoutheastの方角

Within a ruby[p281] element that does not have a ruby[p281] 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[p155] and rp[p288] 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[p287] 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[p281] element represents[p148] 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][p1582]

At any particular time, the segmentation and categorization of content of a ruby[p281] 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[p281] element for which the algorithm is being run.
4. If *root* has a ruby[p281] 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 *saved start index* be null.
9. Let *current base text* be null.
10. *Start mode*: If *index* is greater than or equal to the number of child nodes in *current parent*, then jump to the step labeled *end mode*.
11. If the *index*th node in *current parent* is an `rt`[p287] or `rp`[p288] element, jump to the step labeled *annotation mode*.
12. Set *start index* to the value of *index*.
13. *Base mode*: If the *index*th node in *current parent* is a `ruby`[p281] element, and if *current parent* is the same element as *root*, then push a ruby level[p285] and then jump to the step labeled *start mode*.
14. If the *index*th node in *current parent* is an `rt`[p287] or `rp`[p288] element, then set the current base text[p285] and then jump to the step labeled *annotation mode*.
15. Increment *index* by one.
16. *Base mode post-increment*: If *index* is greater than or equal to 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 `rt`[p287] element, then push a ruby annotation[p286] and jump to the step labeled *annotation mode increment*.
19. If the *index*th node in *current parent* is an `rp`[p288] element, jump to the step labeled *annotation mode increment*.
20. If the *index*th node in *current parent* is not a `Text` node, or is a `Text` node that is not inter-element whitespace[p155], 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 `rt`[p287] element or an `rp`[p288] 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 `Text` node, or is a `Text` node that is not inter-element whitespace[p155], then jump to the step labeled *base mode* (without further incrementing *index*, so the inter-element whitespace[p155] 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 *root*, then pop a ruby level[p286] and jump to the step labeled *base mode post-increment*.
28. *End*: Return *base text segments* and *annotation segments*. Any content of the `ruby`[p281] 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 *text 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:

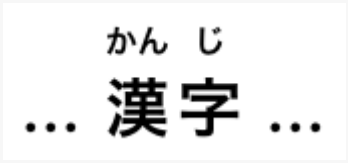
1. Let *rt* be the rt[p287] element that is the *index*th node of *current parent*.
2. Let *annotation range* be a DOM range whose start is the boundary point (*current parent*, *index*) and whose end is the boundary point (*current parent*, *index* plus one) (i.e. that contains only *rt*).
3. Let *new 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:



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:

Example

In this example, each ideograph in the simplified Chinese text 汉字 is annotated with its pinyin reading.

```
...<ruby>汉<rt>hàn</rt>字<rt>zì</rt></ruby>...
```

This might be rendered as:

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".

```
<ruby>
 <ruby>HT<rt>Hypertext</rt>M<rt>Markup</rt>L<rt>Language</rt></ruby>
 <rt>An abstract language for describing documents and applications
</ruby>
```

✔ MDN

4.5.11 The rt element §[p287]

Categories[p154]:

None.

Contexts in which this element can be used[p154]:

As a child of a ruby[p281] element.

Content model[p154]:

Phrasing content[p157].

Tag omission in text/html[p154]:

An rt[p287] element's end tag[p1359] can be omitted if the rt[p287] element is immediately followed by an rt[p287] or rp[p288] element, or if there is no more content in the parent element.

Content attributes[p154]:

Global attributes[p162]

Accessibility considerations[p154]:

For authors.
For implementers.

Sanitization[p154]:

Default[p1249].

DOM interface[p154]:

Uses HTMLElement[p149].

The rt[p287] element marks the ruby text component of a ruby annotation. When it is the child of a ruby[p281] element, it doesn't represent[p148] anything itself, but the ruby[p281] element uses it as part of determining what *it* represents[p148].

An rt[p287] element that is not a child of a ruby[p281] element represents[p148] the same thing as its children.

4.5.12 The rp element §[p288]

Categories[p154]:

None.

Contexts in which this element can be used[p154]:

As a child of a ruby[p281] element, either immediately before or immediately after an rt[p287] element.

Content model[p154]:

Text[p158].

Tag omission in text/html[p154]:

An rp[p288] element's end tag[p1359] can be omitted if the rp[p288] element is immediately followed by an rt[p287] or rp[p288] element, or if there is no more content in the parent element.

Content attributes[p154]:

Global attributes[p162]

Accessibility considerations[p154]:

For authors.
For implementers.

Sanitization[p154]:

Default[p1249].

DOM interface[p154]:

Uses HTMLElement[p149].

The rp[p288] 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[p288] element that is a child of a ruby[p281] element represents[p148] nothing. An rp[p288] element whose parent element is not a ruby[p281] element represents[p148] 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[p288] so that in legacy user agents the readings are in parentheses:

```
...
<ruby>漢<rp>(</rp><rt>かん</rt><rp>)</rp>字<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:

... 漢(かん)字(じ)...

Example

When there are multiple annotations for a segment, rp[p288] 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[p288] elements as well:

```
<ruby>
♥<rp>: </rp><rt>Heart</rt><rp>, </rp><rt lang=fr>Cœur</rt><rp>.</rp>
☘<rp>: </rp><rt>Shamrock</rt><rp>, </rp><rt lang=fr>Trèfle</rt><rp>.</rp>
⋆<rp>: </rp><rt>Star</rt><rp>, </rp><rt lang=fr>Étoile</rt><rp>.</rp>
</ruby>
```

This would make the example render as follows in non-ruby-capable user agents:

♥: Heart, Cœur. ☘: Shamrock, Trèfle. ⋆: Star, Étoile.

4.5.13 The data element §[p289]

Categories[p154]:

Flow content[p157].
Phrasing content[p157].
Palpable content[p158].

Contexts in which this element can be used[p154]:

Where phrasing content[p157] is expected.

Content model[p154]:

Phrasing content[p157].

Tag omission in text/html[p154]:

Neither tag is omissible.

Content attributes[p154]:

Global attributes[p162]
value[p289] — Machine-readable value

Accessibility considerations[p154]:

For authors.
For implementers.

Sanitization[p154]:

Default[p1249] with attributes value[p289].

DOM interface[p154]:

IDL

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

  [CEReactions, Reflect] attribute DOMString value;
};
```

The data[p289] element represents[p148] its contents, along with a machine-readable form of those contents in the value[p289] 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[p290] element can be used instead.

The element can be used for several purposes.

When combined with microformats or the microdata attributes[p826] 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[p289] 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-*[p171] attributes can also be useful in such situations.)

Example

Here, a short table has its numeric values encoded using the data[p289] 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.

```
<script src="sortable.js"></script>
<table class="sortable">
 <thead> <tr> <th> Game <th> Corporations <th> Map Size
 <tbody>
```

```
  <tr> <td> 1830 <td> <data value="8">Eight</data> <td> <data value="93">19+74 hexes (93
total)</data>
  <tr> <td> 1856 <td> <data value="11">Eleven</data> <td> <data value="99">12+87 hexes (99
total)</data>
  <tr> <td> 1870 <td> <data value="10">Ten</data> <td> <data value="149">4+145 hexes (149
total)</data>
</table>
```

4.5.14 The time element §[p290]

Categories[p154]:

Flow content[p157].
Phrasing content[p157].
Palpable content[p158].

Contexts in which this element can be used[p154]:

Where phrasing content[p157] is expected.

Content model[p154]:

If the element has a datetime[p290] attribute: Phrasing content[p157].
Otherwise: Text[p158], but must match requirements described in prose below.

Tag omission in text/html[p154]:

Neither tag is omissible.

Content attributes[p154]:

Global attributes[p162]
datetime[p290] — Machine-readable value

Accessibility considerations[p154]:

For authors.
For implementers.

Sanitization[p154]:

Default[p1249] with attributes datetime[p290].

DOM interface[p154]:

IDL
```
[Exposed=Window]
interface HTMLTimeElement : HTMLElement {
  [HTMLConstructor] constructor();

  [CEReactions, Reflect] attribute DOMString dateTime;
};
```

The time[p290] element represents[p148] its contents, along with a machine-readable form of those contents in the datetime[p290] 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[p290] element that does not have a datetime[p290] content attribute must not have any element descendants.

The **datetime value** of a time[p290] element is the value of the element's datetime[p290] content attribute, if it has one, otherwise the child text content of the time[p290] element.

The datetime value[p290] of a time[p290] element must match one of the following syntaxes.

A valid month string[p86]

Example

```
<time>2011-11</time>
```

A valid date string[p87]

Example

```
<time>2011-11-18</time>
```

A valid yearless date string[p88]

Example

```
<time>11-18</time>
```

A valid time string[p88]

Example

```
<time>14:54</time>
```

Example

```
<time>14:54:39</time>
```

Example

```
<time>14:54:39.929</time>
```

A valid local date and time string[p89]

Example

```
<time>2011-11-18T14:54</time>
```

Example

```
<time>2011-11-18T14:54:39</time>
```

Example

```
<time>2011-11-18T14:54:39.929</time>
```

Example

```
<time>2011-11-18 14:54</time>
```

Example

```
<time>2011-11-18 14:54:39</time>
```

Example

```
<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[p92] is likely more useful.

A valid time-zone offset string[p90]

Example

```
<time>Z</time>
```

Example

```
<time>+0000</time>
```

Example

```
<time>+00:00</time>
```

Example

```
<time>-0800</time>
```

Example

```
<time>-08:00</time>
```

Note

For times without dates (or times referring to events that recur on multiple dates), specifying the geographic location that controls the time is usually more useful than specifying a time zone offset, because geographic locations change time zone offsets with daylight 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][p1587]*

A valid global date and time string[p92]

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>2011-11-18T14:54+00:00</time>
```

Example

```
<time>2011-11-18T14:54:39+00:00</time>
```

Example

```
<time>2011-11-18T14:54:39.929+00:00</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>
```

Example

```
<time>2011-11-18 06:54-08:00</time>
```

Example

```
<time>2011-11-18 06:54:39-08:00</time>
```

Example

```
<time>2011-11-18 06:54:39.929-08:00</time>
```

Note

Times with dates and a time zone offset are useful for specifying specific events, or recurring virtual events where the time is not anchored to a specific geographic location. For example, the precise time of an asteroid impact, or a particular meeting in a series of meetings held at 1400 UTC every day, regardless of whether any particular part of the world is observing daylight saving time or not. For events where the precise time varies by the local time zone offset of a specific geographic location, a valid local date and time string[p89] combined with that geographic location is likely more useful.

A valid week string[p93]

Example

```
<time>2011-W47</time>
```

Four or more ASCII digits, at least one of which is not U+0030 DIGIT ZERO (0)

Example

```
<time>2011</time>
```

Example

```
<time>0001</time>
```

A valid duration string[p94]

Example

```
<time>PT4H18M3S</time>
```

Example

```
<time>4h 18m 3s</time>
```

The **machine-readable equivalent of the element's contents** must be obtained from the element's datetime value[p290] by using the following algorithm:

1. If parsing a month string[p86] from the element's datetime value[p290] returns a month[p86], that is the machine-readable equivalent; return.
2. If parsing a date string[p87] from the element's datetime value[p290] returns a date[p87], that is the machine-readable equivalent; return.
3. If parsing a yearless date string[p88] from the element's datetime value[p290] returns a yearless date[p87], that is the machine-readable equivalent; return.
4. If parsing a time string[p89] from the element's datetime value[p290] returns a time[p88], that is the machine-readable equivalent; return.
5. If parsing a local date and time string[p90] from the element's datetime value[p290] returns a local date and time[p89], that is the machine-readable equivalent; return.
6. If parsing a time-zone offset string[p91] from the element's datetime value[p290] returns a time-zone offset[p90], that is the machine-readable equivalent; return.
7. If parsing a global date and time string[p92] from the element's datetime value[p290] returns a global date and time[p91], that is the machine-readable equivalent; return.
8. If parsing a week string[p93] from the element's datetime value[p290] returns a week[p93], that is the machine-readable equivalent; return.
9. If the element's datetime value[p290] 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[p95] from the element's datetime value[p290] returns a duration[p94], that is the machine-readable equivalent; return.
11. There is no machine-readable equivalent.

Note

The algorithms referenced above are intended to be designed such that for any arbitrary string s, only one of the algorithms returns a value. A more efficient approach might be to create a single algorithm that parses all these data types in one pass; developing such an algorithm is left as an exercise to the reader.

Example

The time[p290] element can be used to encode dates, for example in microformats. The following shows a hypothetical way of

encoding an event using a variant on hCalendar that uses the time element:

```
<div class="vevent">
 <a class="url" href="http://www.web2con.com/">http://www.web2con.com/</a>
 <span class="summary">Web 2.0 Conference</span>:
 <time class="dtstart" datetime="2005-10-05">October 5</time> -
 <time class="dtend" datetime="2005-10-07">7</time>,
 at the <span class="location">Argent Hotel, San Francisco, CA</span>
</div>
```

Example

Here, a fictional microdata vocabulary based on the Atom vocabulary is used with the time element to mark up a blog post's publication date.

```
<article 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:

```
<article itemscope itemtype="http://schema.org/BlogPosting">
 <h1 itemprop="headline">Small tasks</h1>
 <footer>Published <time itemprop="datePublished" datetime="2009-08-30">yesterday</time>.</footer>
 <p itemprop="articleBody">I put a bike bell on her bike.</p>
</article>
```

Example

In the following snippet, the time element is used to encode a date in the ISO8601 format, for later processing by a script:

```
<p>Our first date was <time datetime="2006-09-23">a Saturday</time>.</p>
```

In this second snippet, the value includes a time:

```
<p>We stopped talking at <time datetime="2006-09-24T05:00-07:00">5am the next morning</time>.</p>
```

A script loaded by the page (and thus privy to the page's internal convention of marking up dates and times using the time element) could scan through the page and look at all the time elements therein to create an index of dates and times.

Example

For example, this element conveys the string "Friday" with the additional semantic that the 18th of November 2011 is the meaning that corresponds to "Friday":

```
Today is <time datetime="2011-11-18">Friday</time>.
```

Example

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 §[p297]

Categories[p154]:

Flow content[p157].
Phrasing content[p157].
Palpable content[p158].

Contexts in which this element can be used[p154]:

Where phrasing content[p157] is expected.

Content model[p154]:

Phrasing content[p157].

Tag omission in text/html[p154]:

Neither tag is omissible.

Content attributes[p154]:

Global attributes[p162]

Accessibility considerations[p154]:

For authors.
For implementers.

Sanitization[p154]:

Default[p1249].

DOM interface[p154]:

Uses `HTMLElement`[p149].

The `code`[p297] element represents[p148] a fragment of computer code. This could be an XML element name, a filename, a computer program, or any other string that a computer would recognize.

There is no formal way to indicate the language of computer code being marked up. Authors who wish to mark `code`[p297] elements with the language used, e.g. so that syntax highlighting scripts can use the right rules, can use the `class`[p162] 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.

```
<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`[p242] and `code`[p297] elements.

```
<pre><code class="language-pascal">var i: Integer;
begin
   i := 1;
end.</code></pre>
```

A class is used in that example to indicate the language used.

Note

See the pre[p242] element for more details.

4.5.16 The var element §[p298]

Categories[p154]:

Flow content[p157].
Phrasing content[p157].
Palpable content[p158].

Contexts in which this element can be used[p154]:

Where phrasing content[p157] is expected.

Content model[p154]:

Phrasing content[p157].

Tag omission in text/html[p154]:

Neither tag is omissible.

Content attributes[p154]:

Global attributes[p162]

Accessibility considerations[p154]:

For authors.
For implementers.

Sanitization[p154]:

Default[p1249].

DOM interface[p154]:

Uses HTMLElement[p149].

The var[p298] element represents[p148] 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:

```
<p>If there are <var>n</var> pipes leading to the ice
cream factory then I expect at <em>least</em> <var>n</var>
flavors of ice cream to be available for purchase!</p>
```

For mathematics, in particular for anything beyond the simplest of expressions, MathML is more appropriate. However, the var[p298] 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[p298].

```
<figure>
 <math>
  <mi>a</mi>
  <mo>=</mo>
  <msqrt>
   <msup><mi>b</mi><mn>2</mn></msup>
   <mi>+</mi>
   <msup><mi>c</mi><mn>2</mn></msup>
  </msqrt>
```

```
 </math>
 <figcaption>
  Using Pythagoras' theorem to solve for the hypotenuse <var>a</var> of
  a triangle with sides <var>b</var> and <var>c</var>
 </figcaption>
</figure>
```

Example

Here, the equation describing mass-energy equivalence is used in a sentence, and the var[p298] element is used to mark the variables and constants in that equation:

```
<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> <var>c</var><sup>2</sup>. The teacher
looked pleased.</p>
```

4.5.17 The samp element §[p299]

Categories[p154]:

Flow content[p157].
Phrasing content[p157].
Palpable content[p158].

Contexts in which this element can be used[p154]:

Where phrasing content[p157] is expected.

Content model[p154]:

Phrasing content[p157].

Tag omission in text/html[p154]:

Neither tag is omissible.

Content attributes[p154]:

Global attributes[p162]

Accessibility considerations[p154]:

For authors.
For implementers.

Sanitization[p154]:

Default[p1249].

DOM interface[p154]:

Uses HTMLElement[p149].

The samp[p299] element represents[p148] sample or quoted output from another program or computing system.

Note

See the pre[p242] and kbd[p300] elements for more details.

Note

This element can be contrasted with the output[p611] element, which can be used to provide immediate output in a web application.

Example

This example shows the samp[p299] element being used inline:

```
<p>The computer said <samp>Too much cheese in tray
```

```
two</samp> but I didn't know what that meant.</p>
```

Example

This second example shows a block of sample output from a console program. Nested samp[p299] and kbd[p300] 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[p299] that are annotated with even more detailed markup, to enable very precise styling. To achieve this, span[p309] elements are used.

```
<pre><samp><span class="prompt">jdoe@mowmow:~$</span> <kbd>ssh demo.example.com</kbd>
Last login: Tue Apr 12 09:10:17 2005 from mowmow.example.com on pts/1
Linux demo 2.6.10-grsec+gg3+e+fhs6b+nfs+gr0501+++p3+c4a+gr2b-reslog-v6.189 #1 SMP Tue Feb 1
11:22:36 PST 2005 i686 unknown

<span class="prompt">jdoe@demo:~$</span> <span class="cursor">_</span></samp></pre>
```

Example

This third example shows a block of input and its respective output. The example uses both code[p297] and samp[p299] elements.

```
<pre>
<code class="language-javascript">console.log(2.3 + 2.4)</code>
<samp>4.699999999999999</samp>
</pre>
```

✔ MDN

4.5.18 The kbd element §[p300]

Categories[p154]:

Flow content[p157].
Phrasing content[p157].
Palpable content[p158].

Contexts in which this element can be used[p154]:

Where phrasing content[p157] is expected.

Content model[p154]:

Phrasing content[p157].

Tag omission in text/html[p154]:

Neither tag is omissible.

Content attributes[p154]:

Global attributes[p162]

Accessibility considerations[p154]:

For authors.
For implementers.

Sanitization[p154]:

Default[p1249].

DOM interface[p154]:

Uses HTMLElement[p149].

The kbd[p300] element represents[p148] user input (typically keyboard input, although it may also be used to represent other input, such as voice commands).

When the kbd[p300] element is nested inside a samp[p299] element, it represents the input as it was echoed by the system.

When the kbd[p300] element *contains* a samp[p299] 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>Shift</kbd> + <kbd>F3</kbd></kbd></p>
```

In this second example, the user is told to pick a particular menu item. The outer kbd element marks up a block of input, with the inner kbd elements representing each individual step of the input, and the samp elements inside them indicating that the steps are input based on something being displayed by the system, in this case menu labels:

```
<p>To make George eat an apple, select
    <kbd><kbd><samp>File</samp></kbd>|<kbd><samp>Eat Apple...</samp></kbd></kbd>
</p>
```

Such precision isn't necessary; the following is equally fine:

```
<p>To make George eat an apple, select <kbd>File | Eat Apple...</kbd></p>
```

4.5.19 The sub and sup elements

Categories:

Flow content.
Phrasing content.
Palpable content.

Contexts in which this element can be used:

Where phrasing content is expected.

Content model:

Phrasing content.

Tag omission in text/html:

Neither tag is omissible.

Content attributes:

Global attributes

Accessibility considerations:

The sub element: for authors; for implementers.
The sup element: for authors; for implementers.

Sanitization:

Default.

DOM interface:

Use HTMLElement.

The sup element represents a superscript and the sub element represents a subscript.

These elements must be used only to mark up typographical conventions with specific meanings, not for typographical presentation for presentation's sake. For example, it would be inappropriate for the sub and sup elements to be used in the name of the LaTeX document preparation system. In general, authors should use these elements only if the *absence* of those elements would change the meaning of the content.

In certain languages, superscripts are part of the typographical conventions for some abbreviations.

Example

```
<p>Their names are
<span lang="fr"><abbr>M<sup>lle</sup></abbr> Gwendoline</span> and
<span lang="fr"><abbr>M<sup>me</sup></abbr> Denise</span>.</p>
```

The sub[p301] element can be used inside a var[p298] element, for variables that have subscripts.

Example

Here, the sub[p301] element is used to represent the subscript that identifies the variable in a family of variables:

```
<p>The coordinate of the <var>i</var>th point is
(<var>x<sub><var>i</var></sub></var>, <var>y<sub><var>i</var></sub></var>).
For example, the 10th point has coordinate
(<var>x<sub>10</sub></var>, <var>y<sub>10</sub></var>).</p>
```

Mathematical expressions often use subscripts and superscripts. Authors are encouraged to use MathML for marking up mathematics, but authors may opt to use sub[p301] and sup[p301] if detailed mathematical markup is not desired. [MATHML][p1584]

Example

```
<var>E</var>=<var>m</var><var>c</var><sup>2</sup>
```

```
f(<var>x</var>, <var>n</var>) = log<sub>4</sub><var>x</var><sup><var>n</var></sup>
```

✔ MDN

4.5.20 The i element §[p302]

Categories[p154]:

Flow content[p157].
Phrasing content[p157].
Palpable content[p158].

Contexts in which this element can be used[p154]:

Where phrasing content[p157] is expected.

Content model[p154]:

Phrasing content[p157].

Tag omission in text/html[p154]:

Neither tag is omissible.

Content attributes[p154]:

Global attributes[p162]

Accessibility considerations[p154]:

For authors.
For implementers.

Sanitization[p154]:

Default[p1249].

DOM interface[p154]:

Uses HTMLElement[p149].

The i[p302] element represents[p148] 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[p165] attributes (or, in XML, lang attributes in the XML namespace).

Example

The examples below show uses of the i element:

```
<p>The <i class="taxonomy">Felis silvestris catus</i> is cute.</p>
<p>The term <i>prose content</i> is defined above.</p>
<p>There is a certain <i lang="fr">je ne sais quoi</i> in the air.</p>
```

In the following example, a dream sequence is marked up using i elements.

```
<p>Raymond tried to sleep.</p>
<p><i>The ship sailed away on Thursday</i>, he
dreamt. <i>The ship had many people aboard, including a beautiful
princess called Carey. He watched her, day-in, day-out, hoping she
would notice him, but she never did.</i></p>
<p><i>Finally one night he picked up the courage to speak with
her—</i></p>
<p>Raymond woke with a start as the fire alarm rang out.</p>
```

Authors can use the class attribute on the i element to identify why the element is being used, so that if the style of a particular use (e.g. dream sequences as opposed to taxonomic terms) is to be changed at a later date, the author doesn't have to go through the entire document (or series of related documents) annotating each use.

Authors are encouraged to consider whether other elements might be more applicable than the i element, for instance the em element for marking up stress emphasis, or the dfn element to mark up the defining instance of a term.

Note

Style sheets can be used to format i elements, just like any other element can be restyled. Thus, it is not the case that content in i elements will necessarily be italicized.

4.5.21 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.

Content attributes:

Global attributes

Accessibility considerations:

For authors.
For implementers.

Sanitization:

Default.

DOM interface:

Uses HTMLElement.

The b element represents a span of text to which attention is being drawn for utilitarian purposes without conveying any extra importance and with no implication of an alternate voice or mood, such as key words in a document abstract, product names in a

review, actionable words in interactive text-driven software, or an article lede.

Example

The following example shows a use of the b[p303] element to highlight key words without marking them up as important:

```
<p>The <b>frobonitor</b> and <b>barbinator</b> components are fried.</p>
```

Example

In the following example, objects in a text adventure are highlighted as being special by use of the b[p303] element.

```
<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[p303] element is appropriate is in marking up the lede (or lead) sentence or paragraph. The following example shows how a BBC article about kittens adopting a rabbit as their own could be marked up:

```
<article>
 <h2>Kittens 'adopted' by pet rabbit</h2>
 <p><b class="lede">Six abandoned kittens have found an
 unexpected new mother figure – a pet rabbit.</b></p>
 <p>Veterinary nurse Melanie Humble took the three-week-old
 kittens to her Aberdeen home.</p>
[...]
```

As with the i[p302] element, authors can use the class[p162] attribute on the b[p303] 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[p303] element should be used as a last resort when no other element is more appropriate. In particular, headings should use the h1[p224] to h6[p224] elements, stress emphasis should use the em[p270] element, importance should be denoted with the strong[p271] element, and text marked or highlighted should use the mark[p305] element.

Example

The following would be *incorrect* usage:

```
<p><b>WARNING!</b> Do not frob the barbinator!</p>
```

In the previous example, the correct element to use would have been strong[p271], not b[p303].

Note

Style sheets can be used to format b[p303] elements, just like any other element can be restyled. Thus, it is not the case that content in b[p303] elements will necessarily be boldened.

4.5.22 The u element §[p304]

Categories[p154]:

Flow content[p157].
Phrasing content[p157].
Palpable content[p158].

Contexts in which this element can be used[p154]:

Where phrasing content[p157] is expected.

Content model[p154]:

Phrasing content[p157].

Tag omission in text/html[p154]:

Neither tag is omissible.

Content attributes[p154]:

Global attributes[p162]

Accessibility considerations[p154]:

For authors.
For implementers.

Sanitization[p154]:

Default[p1249].

DOM interface[p154]:

Uses `HTMLElement`[p149].

The `u`[p304] element represents[p148] 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`[p270] element should be used; for marking key words or phrases either the `b`[p303] element or the `mark`[p305] element should be used, depending on the context; for marking book titles, the `cite`[p275] element should be used; for labeling text with explicit textual annotations, the `ruby`[p281] element should be used; for technical terms, taxonomic designation, transliteration, a thought, or for labeling ship names in Western texts, the `i`[p302] element should be used.

Note

The default rendering of the `u`[p304] element in visual presentations clashes with the conventional rendering of hyperlinks (underlining). Authors are encouraged to avoid using the `u`[p304] element where it could be confused for a hyperlink.

Example

In this example, a `u`[p304] element is used to mark a word as misspelt:

```
<p>The <u>see</u> is full of fish.</p>
```

4.5.23 The `mark` element §[p305]

Categories[p154]:

Flow content[p157].
Phrasing content[p157].
Palpable content[p158].

Contexts in which this element can be used[p154]:

Where phrasing content[p157] is expected.

Content model[p154]:

Phrasing content[p157].

Tag omission in text/html[p154]:

Neither tag is omissible.

Content attributes[p154]:

Global attributes[p162]

Accessibility considerations[p154]:

For authors.
For implementers.

Sanitization[p154]:

Default[p1249].

DOM interface[p154]:

Uses HTMLElement[p149].

The mark[p305] element represents[p148] a run of text in one document marked or highlighted for reference[p148] 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[p305] element can be used to bring attention to a particular part of a quotation:

```
<p lang="en-US">Consider the following quote:</p>
<blockquote lang="en-GB">
 <p>Look around and you will find, no-one's really
 <mark>colour</mark> blind.</p>
</blockquote>
<p lang="en-US">As we can tell from the <em>spelling</em> of the word,
the person writing this quote is clearly not American.</p>
```

(If the goal was to mark the element as misspelt, however, the u[p304] element, possibly with a class, would be more appropriate.)

Example

Another example of the mark[p305] element is highlighting parts of a document that are matching some search string. If someone looked at a document, and the server knew that the user was searching for the word "kitten", then the server might return the document with one paragraph modified as follows:

```
<p>I also have some <mark>kitten</mark>s who are visiting me
these days. They're really cute. I think they like my garden! Maybe I
should adopt a <mark>kitten</mark>.</p>
```

Example

In the following snippet, a paragraph of text refers to a specific part of a code fragment.

```
<p>The highlighted part below is where the error lies:</p>
<pre><code>var i: Integer;
begin
   i := <mark>1.1</mark>;
end.</code></pre>
```

This is separate from *syntax highlighting*, for which span[p309] is more appropriate. Combining both, one would get:

```
<p>The highlighted part below is where the error lies:</p>
<pre><code><span class=keyword>var</span> <span class=ident>i</span>: <span
class=type>Integer</span>;
<span class=keyword>begin</span>
   <span class=ident>i</span> := <span class=literal><mark>1.1</mark></span>;
<span class=keyword>end</span>.</code></pre>
```

Example

This is another example showing the use of mark[p305] 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[p305] elements in quotes to render in italics.

```
<style>
 blockquote mark, q mark {
   font: inherit; font-style: italic;
```

```
    text-decoration: none;
    background: transparent; color: inherit;
  }
  .bubble em {
    font: inherit; font-size: larger;
    text-decoration: underline;
  }
</style>
<article>
 <h1>She knew</h1>
 <p>Did you notice the subtle joke in the joke on panel 4?</p>
 <blockquote>
  <p class="bubble">I didn't <em>want</em> to believe. <mark>Of course
  on some level I realized it was a known-plaintext attack.</mark> But I
  couldn't admit it until I saw for myself.</p>
 </blockquote>
 <p>(Emphasis mine.) I thought that was great. It's so pedantic, yet it
 explains everything neatly.</p>
</article>
```

Note, incidentally, the distinction between the em[p270] element in this example, which is part of the original text being quoted, and the mark[p305] 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[p271]) as opposed to denoting the *relevance* of a span of text (mark[p305]). It is an extract from a textbook, where the extract has had the parts relevant to the exam highlighted. The safety warnings, important though they may be, are apparently not relevant to the exam.

```
<h3>Wormhole Physics Introduction</h3>

<p><mark>A wormhole in normal conditions can be held open for a
maximum of just under 39 minutes.</mark> Conditions that can increase
the time include a powerful energy source coupled to one or both of
the gates connecting the wormhole, and a large gravity well (such as a
black hole).</p>

<p><mark>Momentum is preserved across the wormhole. Electromagnetic
radiation can travel in both directions through a wormhole,
but matter cannot.</mark></p>

<p>When a wormhole is created, a vortex normally forms.
<strong>Warning: The vortex caused by the wormhole opening will
annihilate anything in its path.</strong> Vortexes can be avoided when
using sufficiently advanced dialing technology.</p>

<p><mark>An obstruction in a gate will prevent it from accepting a
wormhole connection.</mark></p>
```

4.5.24 The bdi element §[p307]

Categories[p154]:

Flow content[p157].
Phrasing content[p157].
Palpable content[p158].

Contexts in which this element can be used[p154]:

Where phrasing content[p157] is expected.

Content model[p154]**:**

Phrasing content[p157].

Tag omission in text/html[p154]**:**

Neither tag is omissible.

Content attributes[p154]**:**

Global attributes[p162]
Also, the dir[p167] global attribute has special semantics on this element.

Accessibility considerations[p154]**:**

For authors.
For implementers.

Sanitization[p154]**:**

Default[p1249].

DOM interface[p154]**:**

Uses HTMLElement[p149].

The bdi[p307] element represents[p148] a span of text that is to be isolated from its surroundings for the purposes of bidirectional text formatting. [BIDI][p1579]

Note

The dir[p167] global attribute defaults to auto[p167] on this element (it never inherits from the parent element like with other elements).

Note

This element has rendering requirements involving the bidirectional algorithm[p177].

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[p307] element were not used, the username of the Arabic user would end up confusing the text (the bidirectional algorithm would put the colon and the number "3" next to the word "User" rather than next to the word "posts").

```
<ul>
 <li>User <bdi>jcranmer</bdi>: 12 posts.
 <li>User <bdi>hober</bdi>: 5 posts.
 <li>User <bdi>إيان</bdi>: 3 posts.
</ul>
```

- User jcranmer: 12 posts.
- User hober: 5 posts.
- User إيان: 3 posts.

When using the bdi[p307] element, the username acts as expected.

- User **jcranmer**: 12 posts.
- User **hober**: 5 posts.
- User 3 :**إيان** posts.

If the bdi[p307] element were to be replaced by a b[p303] element, the username would confuse the bidirectional algorithm and the third bullet would end up saying "User 3 :", followed by the Arabic name (right-to-left), followed by "posts" and a period.

4.5.25 The bdo element §[p309]

Categories[p154]:

Flow content[p157].
Phrasing content[p157].
Palpable content[p158].

Contexts in which this element can be used[p154]:

Where phrasing content[p157] is expected.

Content model[p154]:

Phrasing content[p157].

Tag omission in text/html[p154]:

Neither tag is omissible.

Content attributes[p154]:

Global attributes[p162]
Also, the dir[p167] global attribute has special semantics on this element.

Accessibility considerations[p154]:

For authors.
For implementers.

Sanitization[p154]:

Default[p1249].

DOM interface[p154]:

Uses HTMLElement[p149].

The bdo[p309] element represents[p148] explicit text directionality formatting control for its children. It allows authors to override the Unicode bidirectional algorithm by explicitly specifying a direction override. [BIDI][p1579]

Authors must specify the dir[p167] attribute on this element, with the value ltr[p167] to specify a left-to-right override and with the value rtl[p167] to specify a right-to-left override. The auto[p167] value must not be specified.

Note

This element has rendering requirements involving the bidirectional algorithm[p177].

4.5.26 The span element §[p309]

Categories[p154]:

Flow content[p157].
Phrasing content[p157].
Palpable content[p158].

Contexts in which this element can be used[p154]:

Where phrasing content[p157] is expected.

Content model[p154]:

Phrasing content[p157].

Tag omission in text/html[p154]:

Neither tag is omissible.

Content attributes[p154]:

Global attributes[p162]

Accessibility considerations[p154]:

For authors.
For implementers.

Sanitization[p154]**:**

Default[p1249].

DOM interface[p154]**:**

IDL

```
[Exposed=Window]
interface HTMLSpanElement : HTMLElement {
  [HTMLConstructor] constructor();
};
```

The span[p309] element doesn't mean anything on its own, but can be useful when used together with the global attributes[p162], e.g. class[p162], lang[p165], or dir[p167]. It represents[p148] its children.

Example

In this example, a code fragment is marked up using span[p309] elements and class[p162] attributes so that its keywords and identifiers can be color-coded from CSS:

```
<pre><code class="lang-c"><span class="keyword">for</span> (<span class="ident">j</span> = 0;
<span class="ident">j</span> &lt; 256; <span class="ident">j</span>++) {
  <span class="ident">i_t3</span> = (<span class="ident">i_t3</span> & 0x1ffff) | (<span
class="ident">j</span> &lt;&lt; 17);
  <span class="ident">i_t6</span> = (((((((<span class="ident">i_t3</span> >> 3) ^ <span
class="ident">i_t3</span>) >> 1) ^ <span class="ident">i_t3</span>) >> 8) ^ <span
class="ident">i_t3</span>) >> 5) & 0xff;
  <span class="keyword">if</span> (<span class="ident">i_t6</span> == <span
class="ident">i_t1</span>)
    <span class="keyword">break</span>;
}</code></pre>
```

4.5.27 The br element §[p310]

Categories[p154]**:**

Flow content[p157].
Phrasing content[p157].

Contexts in which this element can be used[p154]**:**

Where phrasing content[p157] is expected.

Content model[p154]**:**

Nothing[p155].

Tag omission in text/html[p154]**:**

No end tag[p1359].

Content attributes[p154]**:**

Global attributes[p162]

Accessibility considerations[p154]**:**

For authors.
For implementers.

Sanitization[p154]**:**

Default[p1249].

DOM interface[p154]**:**

IDL

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

```
  // also has obsolete members
};
```

The br[p310] element represents[p148] 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[p310] 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[p310] element:

```
<p>P. Sherman<br>
42 Wallaby Way<br>
Sydney</p>
```

br[p310] elements must not be used for separating thematic groups in a paragraph.

Example

The following examples are non-conforming, as they abuse the br[p310] element:

```
<p><a ...>34 comments.</a><br>
<a ...>Add a comment.</a></p>
```

```
<p><label>Name: <input name="name"></label><br>
<label>Address: <input name="address"></label></p>
```

Here are alternatives to the above, which are correct:

```
<p><a ...>34 comments.</a></p>
<p><a ...>Add a comment.</a></p>
```

```
<p><label>Name: <input name="name"></label></p>
<p><label>Address: <input name="address"></label></p>
```

If a paragraph[p159] consists of nothing but a single br[p310] 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[p310] elements must not be considered part of the surrounding text.

Note

This element has rendering requirements involving the bidirectional algorithm[p177].

4.5.28 The wbr element §[p311]

Categories[p154]:

Flow content[p157].
Phrasing content[p157].

Contexts in which this element can be used[p154]:

Where phrasing content[p157] is expected.

Content model[p154]**:**

Nothing[p155].

Tag omission in text/html[p154]**:**

No end tag[p1359].

Content attributes[p154]**:**

Global attributes[p162]

Accessibility considerations[p154]**:**

For authors.
For implementers.

Sanitization[p154]**:**

Default[p1249].

DOM interface[p154]**:**

Uses `HTMLElement`[p149].

The `wbr`[p311] element represents[p148] 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`[p311] element.

```
<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`[p311] elements must not be considered part of the surrounding text.

Example

```
var wbr = document.createElement("wbr");
wbr.textContent = "This is wrong";
document.body.appendChild(wbr);
```

Note

This element has rendering requirements involving the bidirectional algorithm[p177].

4.5.29 Usage summary §[p312]

This section is non-normative.

Element	Purpose	Example
`a`[p267]	Hyperlinks	`Visit my <a href="drinks.html">drinks</a> page.`
`em`[p270]	Stress emphasis	`I must say I <em>adore</em> lemonade.`
`strong`[p271]	Importance	`This tea is <strong>very hot</strong>.`
`small`[p272]	Side comments	`These grapes are made into wine. <small>Alcohol is addictive.</small>`
`s`[p274]	Inaccurate text	`Price: <s>£4.50</s> £2.00!`
`cite`[p275]	Titles of works	`The case <cite>Hugo v. Danielle</cite> is relevant here.`
`q`[p277]	Quotations	`The judge said <q>You can drink water from the fish tank</q> but advised against it.`
`dfn`[p278]	Defining instance	`The term <dfn>organic food</dfn> refers to food produced without synthetic chemicals.`
`abbr`[p279]	Abbreviations	`Organic food in Ireland is certified by the <abbr title="Irish Organic Farmers and`

Element	Purpose	Example
		Growers Association">IOFGA</abbr>.
ruby[p281], rt[p287], rp[p288]	Ruby annotations	<ruby> OJ <rp>(<rt>Orange Juice<rp>)</ruby>
data[p289]	Machine-readable equivalent	Available starting today! <data value="UPC:022014640201">North Coast Organic Apple Cider</data>
time[p290]	Machine-readable equivalent of date- or time-related data	Available starting on <time datetime="2011-11-18">November 18th</time>!
code[p297]	Computer code	The <code>fruitdb</code> program can be used for tracking fruit production.
var[p298]	Variables	If there are <var>n</var> fruit in the bowl, at least <var>n</var>÷2 will be ripe.
samp[p299]	Computer output	The computer said <samp>Unknown error -3</samp>.
kbd[p300]	User input	Hit <kbd>F1</kbd> to continue.
sub[p301]	Subscripts	Water is H₂O.
sup[p301]	Superscripts	The Hydrogen in heavy water is usually ²H.
i[p302]	Alternative voice	Lemonade consists primarily of Citrus limon.
b[p303]	Keywords	Take a lemon and squeeze it with a juicer.
u[p304]	Annotations	The mixture of apple juice and <u class="spelling">eldeflower</u> juice is very pleasant.
mark[p305]	Highlight	Elderflower cordial, with one <mark>part</mark> cordial to ten <mark>part</mark>s water, stands a<mark>part</mark> from the rest.
bdi[p307]	Text directionality isolation	The recommended restaurant is <bdi lang="">My Juice Café (At The Beach)</bdi>.
bdo[p309]	Text directionality formatting	The proposal is to write English, but in reverse order. "Juice" would become "<bdo dir=rtl>Juice</bdo>">
span[p309]	Other	In French we call it sirop de sureau.
br[p310]	Line break	Simply Orange Juice Company Apopka, FL 32703 U.S.A.
wbr[p311]	Line breaking opportunity	www.simply<wbr>orange<wbr>juice.com

4.6 Links §[p31]

4.6.1 Introduction §[p31]

Links are a conceptual construct, created by a[p267], area[p489], form[p532], and link[p184] elements, that represent[p148] a connection between two resources, one of which is the current Document[p135]. There are three 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[p313] have a fetch and process the linked resource[p190] 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[p1063] to those resources, e.g. to visit them in a browser or download them.

Internal resource links

These are links to resources within the current document, used to give those resources special meaning or behavior.

For link[p184] elements with an href[p185] attribute and a rel[p185] attribute, links must be created for the keywords of the rel[p185] attribute, as defined for those keywords in the link types[p326] section.

Similarly, for a[p267] and area[p489] elements with an href[p314] attribute and a rel[p314] attribute, links must be created for the keywords of the rel[p314] attribute as defined for those keywords in the link types[p326] section. Unlike link[p184] elements, however, a[p267] and area[p489] elements with an href[p314] attribute that either do not have a rel[p314] attribute, or whose rel[p314] attribute has no keywords that are defined as specifying hyperlinks[p313], must also create a hyperlink[p313]. This implied hyperlink has no special meaning (it has no link type[p326]) beyond linking the element's node document to the resource given by the element's href[p314] attribute.

Similarly, for form[p532] elements with a rel[p534] attribute, links must be created for the keywords of the rel[p534] attribute as defined for those keywords in the link types[p326] section. form[p532] elements that do not have a rel[p534] attribute, or whose rel[p534] attribute has no keywords that are defined as specifying hyperlinks[p313], must also create a hyperlink[p313].

A hyperlink[p313] can have one or more **hyperlink annotations** that modify the processing semantics of that hyperlink.

4.6.2 Links created by a[p267] and area[p489] elements §[p314]

The **href** attribute on a[p267] and area[p489] elements must have a value that is a valid URL potentially surrounded by spaces[p100].

Note

The href[p314] attribute on a[p267] and area[p489] elements is not required; when those elements do not have href[p314] attributes they do not create hyperlinks.

The **target** attribute, if present, must be a valid navigable target name or keyword[p1043]. It gives the name of the navigable[p1035] that will be used. User agents use this name when following hyperlinks[p321].

The **download** attribute, if present, indicates that the author intends the hyperlink to be used for downloading a resource[p322]. The attribute may have a value; the value, if any, specifies the default filename 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 filenames, and user agents are likely to adjust filenames 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[p99], each of which must be a valid non-empty URL[p100] whose scheme is an HTTP(S) scheme. The value is used by the user agent for hyperlink auditing[p325].

The **rel** attribute on a[p267] and area[p489] elements controls what kinds of links the elements create. The attribute's value must be an unordered set of unique space-separated tokens[p99]. The allowed keywords and their meanings[p326] are defined below.

rel[p314]'s supported tokens are the keywords defined in HTML link types[p326] which are allowed on a[p267] and area[p489] elements, impact the processing model, and are supported by the user agent. The possible supported tokens are noreferrer[p338], noopener[p337], and opener[p338]. rel[p314]'s supported tokens must only include the tokens from this list that the user agent implements the processing model for.

The rel[p314] 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 **referrerpolicy** attribute is a referrer policy attribute[p104]. Its purpose is to set the referrer policy used when following hyperlinks[p321]. [REFERRERPOLICY][p1585]

When an a[p267] or area[p489] 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[p1063] 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[p314] attribute, and should be to download the specified resource if it does.

The activation behavior of an a[p267] or area[p489] element *element* given an event *event* is:

1. If *element* has no href[p314] attribute, then return.
2. Let *hyperlinkSuffix* be null.
3. If *element* is an a[p267] element, and *event*'s target is an img[p359] with an ismap[p363] attribute specified:

1. Let *x* and *y* be 0.
2. If *event*'s `isTrusted` attribute is initialized to true, 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 0.
4. If *y* is negative, set *y* to 0.
5. Set *hyperlinkSuffix* to the concatenation of U+003F (?), the value of *x* expressed as a base-ten integer using ASCII digits, U+002C (,), and the value of *y* expressed as a base-ten integer using ASCII digits.

4. Let *userInvolvement* be *event*'s user navigation involvement[p1062].
5. If the user has expressed a preference to download the hyperlink, then set *userInvolvement* to "`browser UI`[p1062]".

Note

That is, if the user has expressed a specific preference for downloading, this no longer counts as merely "`activation`[p1062]".

6. If *element* has a `download`[p314] attribute, or if the user has expressed a preference to download the hyperlink, then download the hyperlink[p322] created by *element* with *hyperlinkSuffix*[p322] set to *hyperlinkSuffix* and *userInvolvement*[p322] set to *userInvolvement*.
7. Otherwise, follow the hyperlink[p321] created by *element* with *hyperlinkSuffix*[p321] set to *hyperlinkSuffix* and *userInvolvement*[p321] set to *userInvolvement*.

4.6.3 API for hyperlink elements §[p315]

IDL

```
interface mixin HyperlinkElementUtils {
  readonly attribute USVString origin;
  [CEReactions] attribute USVString protocol;
  [CEReactions] attribute USVString username;
  [CEReactions] attribute USVString password;
  [CEReactions] attribute USVString host;
  [CEReactions] attribute USVString hostname;
  [CEReactions] attribute USVString port;
  [CEReactions] attribute USVString pathname;
  [CEReactions] attribute USVString search;
  [CEReactions] attribute USVString hash;

  [CEReactions, Reflect] attribute DOMString hreflang;
  [CEReactions, Reflect] attribute DOMString type;
};
```

For web developers (non-normative)

***hyperlink*.origin**[p316]

Returns the hyperlink's URL's origin.

***hyperlink*.protocol**[p317]

Returns the hyperlink's URL's scheme.

Can be set, to change the URL's scheme.

***hyperlink*.username**[p317]

Returns the hyperlink's URL's username.

Can be set, to change the URL's username.

***hyperlink*.password**[p317]

Returns the hyperlink's URL's password.

Can be set, to change the URL's password.

***hyperlink*.host**[p317]

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**[p318]

Returns the hyperlink's URL's host.

Can be set, to change the URL's host.

***hyperlink*.port**[p318]

Returns the hyperlink's URL's port.

Can be set, to change the URL's port.

***hyperlink*.pathname**[p318]

Returns the hyperlink's URL's path.

Can be set, to change the URL's path.

***hyperlink*.search**[p319]

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**[p319]

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 HyperlinkElementUtils[p315] mixin is a **hyperlink element**.

A hyperlink element[p316] has an associated **url** (null or a URL). It is initially null.

A hyperlink element[p316] has the following extract an origin[p943] steps:

1. If this's url[p316] is null, then return null.
2. Return this's url[p316]'s origin.

A hyperlink element[p316] must have an associated **set the url** algorithm.

When hyperlink elements[p316] are created, the user agent must set the url[p316].

A hyperlink element[p316] has an associated **reinitialize url** algorithm, which runs these steps:

1. If the element's url[p316] is non-null, its scheme is "`blob`", and it has an opaque path, then terminate these steps.
2. Set the url[p316].

A hyperlink element[p316] must have an associated **update `href`** algorithm.

The **`hreflang`** attribute on hyperlink elements[p316] that create hyperlinks[p313], if present, gives the language of the linked resource. It is purely advisory. The value must be a valid BCP 47 language tag. [BCP47][p1579] 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 on hyperlink elements[p316], 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`[p316] attribute authoritative — upon fetching the resource, user agents must not use metadata included in the link to the resource to determine its type.

The **`origin`** getter steps are:

1. Reinitialize url[p316].
2. If this's url[p316] is null, return the empty string.

3. Return the serialization[p939] of this's url[p316]'s origin.

The **protocol** getter steps are:

1. Reinitialize url[p316].
2. If this's url[p316] is null, return ":".
3. Return this's url[p316]'s scheme, followed by ":".

The protocol[p317] setter steps are:

1. Reinitialize url[p316].
2. If this's url[p316] is null, then return.
3. Basic URL parse the given value, followed by ":", with this's url[p316] 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[p316].

The **username** getter steps are:

1. Reinitialize url[p316].
2. If this's url[p316] is null, return the empty string.
3. Return this's url[p316]'s username.

The username[p317] setter steps are:

1. Reinitialize url[p316].
2. Let *url* be this's url[p316].
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[p316].

The **password** getter steps are:

1. Reinitialize url[p316].
2. Let *url* be this's url[p316].
3. If *url* is null, then return the empty string.
4. Return *url*'s password.

The password[p317] setter steps are:

1. Reinitialize url[p316].
2. Let *url* be this's url[p316].
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[p316].

The **host** getter steps are:

1. Reinitialize url[p316].

2. Let *url* be this's url[p316].
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`[p317] setter steps are:

1. Reinitialize url[p316].
2. Let *url* be this's url[p316].
3. If *url* is null or *url* has an opaque path, then return.
4. Basic URL parse the given value, with *url* as *url* and host state as *state override*.
5. Update `href`[p316].

The **`hostname`** getter steps are:

1. Reinitialize url[p316].
2. Let *url* be this's url[p316].
3. If *url* or *url*'s host is null, return the empty string.
4. Return *url*'s host, serialized.

The `hostname`[p318] setter steps are:

1. Reinitialize url[p316].
2. Let *url* be this's url[p316].
3. If *url* is null or *url* has an opaque path, then return.
4. Basic URL parse the given value, with *url* as *url* and hostname state as *state override*.
5. Update `href`[p316].

The **`port`** getter steps are:

1. Reinitialize url[p316].
2. Let *url* be this's url[p316].
3. If *url* or *url*'s port is null, return the empty string.
4. Return *url*'s port, serialized.

The `port`[p318] setter steps are:

1. Reinitialize url[p316].
2. Let *url* be this's url[p316].
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`[p316].

The **`pathname`** getter steps are:

1. Reinitialize url[p316].
2. Let *url* be this's url[p316].

3. If *url* is null, then return the empty string.
4. Return the result of URL path serializing *url*.

The `pathname`[p318] setter steps are:

1. Reinitialize url[p316].
2. Let *url* be this's url[p316].
3. If *url* is null or *url* has an opaque path, 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`[p316].

The **`search`** getter steps are:

1. Reinitialize url[p316].
2. Let *url* be this's url[p316].
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`[p319] setter steps are:

1. Reinitialize url[p316].
2. Let *url* be this's url[p316].
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 *url* as *url* and query state as *state override*.
6. Update `href`[p316].

The **`hash`** getter steps are:

1. Reinitialize url[p316].
2. Let *url* be this's url[p316].
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`[p319] setter steps are:

1. Reinitialize url[p316].
2. Let *url* be this's url[p316].
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`[p316].

4.6.4 API for `a`[p267] and `area`[p489] elements §[p32]

```
IDL
interface mixin HTMLHyperlinkElementUtils {
  [CEReactions, ReflectSetter] stringifier attribute USVString href;
  [CEReactions, Reflect] attribute DOMString target;
};
```

For web developers (non-normative)

***hyperlink*.toString()**

***hyperlink*.href**[p320]

Returns the hyperlink's URL.

Can be set, to change the URL.

The **`href`** getter steps are:

1. Reinitialize url[p316].
2. Let *url* be this's url[p316].
3. If *url* is null and this has no `href`[p314] content attribute, return the empty string.
4. Otherwise, if *url* is null, return this's `href`[p314] content attribute's value.
5. Return *url*, serialized.

The HTML element insertion steps[p46] for `a`[p267] and `area`[p489] elements, given *insertedNode*, are:

1. If *insertedNode* is not connected, then return.
2. Consider speculative loads[p1128] given *insertedNode*'s node document.

The HTML element removing steps[p46] for `a`[p267] and `area`[p489] elements, given *removedNode*, *isSubtreeRoot*, and *oldAncestor* are:

1. If *oldAncestor* is not connected, then return.
2. Consider speculative loads[p1128] given *oldAncestor*'s node document.

The HTML element moving steps[p46] for `a`[p267] and `area`[p489] elements, given *movedNode*, are:

1. Consider speculative loads[p1128] given *movedNode*'s node document.

The following attribute change steps, given *element*, *localName*, *oldValue*, *value*, and *namespace*, are used for all `a`[p267] and `area`[p489] elements:

1. If *namespace* is not null, then return.
2. If *oldValue* equals *value*, then return.
3. If *localName* is `href`[p314], then set the url[p316] given *element*.

   Note

   This is only observable for `blob:` URLs as parsing them involves a Blob URL Store lookup.

4. If *localName* is `href`[p314], `referrerpolicy`[p314], or `rel`[p314], then consider speculative loads[p1128] given *element*'s node document.

To update href[p316] for an HTMLAnchorElement[p268] or HTMLAreaElement[p489] element, set the element's href[p314] content attribute's value to the element's url[p316], serialized.

The set the url[p316] algorithm for HTMLAnchorElement[p268] and HTMLAreaElement[p489] elements is as follows:

1. Set this element's url[p316] to null.
2. If this element's href[p314] content attribute is absent, then return.
3. Let *url* be the result of encoding-parsing a URL[p101] given this element's href[p314] content attribute's value, relative to this element's node document.
4. If *url* is not failure, then set this element's url[p316] to *url*.

4.6.5 Following hyperlinks §[p32] 1

An element *element* **cannot navigate** if any of the following are true:

- *element*'s node document is not fully active[p1051]; or
- *element* is not an a[p267] element and is not connected.

Note

This is also used by form submission[p658] for the form[p532] element. The exception for a[p267] elements is for compatibility with web content.

To **get an element's noopener**, given an a[p267], area[p489], or form[p532] element *element*, a URL record *url*, and a string *target*, perform the following steps. They return a boolean.

1. If *element*'s link types[p326] include the noopener[p337] or noreferrer[p338] keyword, then return true.
2. If *element*'s link types[p326] do not include the opener[p338] keyword and *target* is an ASCII case-insensitive match for "`_blank`", then return true.
3. If *url*'s blob URL entry is not null:
    1. Let *blobOrigin* be *url*'s blob URL entry's environment's origin[p1144].
    2. Let *topLevelOrigin* be *element*'s relevant settings object[p1151]'s top-level origin[p1144].
    3. If *blobOrigin* is not same site[p941] with *topLevelOrigin*, then return true.
4. Return false.

To **follow the hyperlink** created by an element *subject*, given an optional ***hyperlinkSuffix*** (default null) and an optional ***userInvolvement*** (default "none[p1062]"):

1. If *subject* cannot navigate[p321], then return.
2. Let *targetAttributeValue* be the empty string.
3. If *subject* is an a[p267] or area[p489] element, then set *targetAttributeValue* to the result of getting an element's target[p183] given *subject*.
4. Let *urlRecord* be the result of encoding-parsing a URL[p101] given *subject*'s href[p314] attribute value, relative to *subject*'s node document.
5. If *urlRecord* is failure, then return.
6. Let *noopener* be the result of getting an element's noopener[p321] with *subject*, *urlRecord*, and *targetAttributeValue*.
7. Let *targetNavigable* be the first return value of applying the rules for choosing a navigable[p1044] given *targetAttributeValue*, *subject*'s node navigable[p1036], and *noopener*.
8. If *targetNavigable* is null, then return.

9. Let *urlString* be the result of applying the URL serializer to *urlRecord*.
10. If *hyperlinkSuffix* is non-null, then append it to *urlString*.
11. Navigate[p1063] *targetNavigable* to *urlString* using *subject*'s node document, with *referrerPolicy*[p1063] set to *subject*'s hyperlink referrer policy[p322], *userInvolvement*[p1063] set to *userInvolvement*, and *sourceElement*[p1063] set to *subject*.

> **Note**
>
> *Unlike many other types of navigations, following hyperlinks does not have special "*`replace`[p1062]*" behavior for when documents are not completely loaded*[p1114]*. This is true for both user-initiated instances of following hyperlinks, as well as script-triggered ones via, e.g.,* `aElement.click()`.

The **hyperlink referrer policy** for an element *subject* is the value returned by the following steps:

1. If *subject*'s link types[p326] includes the `noreferrer`[p338] keyword, then return "`no-referrer`".
2. Return the current state of *subject*'s `referrerpolicy`[p314] content attribute.

4.6.6 Downloading resources §[p322]

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`[p314] attribute can be specified on the `a`[p267] or `area`[p489] element that creates the hyperlink[p313] to that resource.

The attribute can furthermore be given a value, to specify the filename 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][p1586]

In cross-origin situations, the `download`[p314] 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.)

To **download the hyperlink** created by an element *subject*, given an optional ***hyperlinkSuffix*** (default null) and an optional ***userInvolvement*** (default "`none`[p1062]"):

1. If *subject* cannot navigate[p321], then return.
2. If *subject*'s node document's active sandboxing flag set[p959] has the sandboxed downloads browsing context flag[p958] set, then return.
3. Let *urlString* be the result of encoding-parsing-and-serializing a URL[p101] given *subject*'s `href`[p314] attribute value, relative to *subject*'s node document.
4. If *urlString* is failure, then return.
5. If *hyperlinkSuffix* is non-null, then append it to *urlString*.
6. If *userInvolvement* is not "`browser UI`[p1062]":
    1. Assert: *subject* has a `download`[p314] attribute.
    2. Let *navigation* be *subject*'s relevant global object[p1151]'s navigation API[p995].
    3. Let *filename* be the value of *subject*'s `download`[p314] attribute.
    4. Let *continue* be the result of firing a download request `navigate` event[p1020] at *navigation* with *destinationURL*[p1020] set to *urlString*, *userInvolvement*[p1020] set to *userInvolvement*, *sourceElement*[p1020] set to *subject*, and *filename*[p1020] set to *filename*.
    5. If *continue* is false, then return.
    6. Inform the navigation API about aborting navigation[p1010] given *subject*'s node navigable[p1036].
7. Optionally, the user agent may return, if it believes doing so would safeguard the user from a potentially hostile download.

8. Let *request* be a new request whose URL is *urlString*, client is entry settings object[p1148], initiator is "download", destination is the empty string, and whose use-URL-credentials flag is set.

9. Fetch *request*, with *processResponse* set to the following steps given response *response*:

    1. Handle as a download[p323] *response* with *subject*'s node navigable[p1036] and null.

To **handle as a download** a response *response* with a navigable[p1035] *navigable* and a navigation ID[p1062] or null *navigationId*:

1. Let *suggestedFilename* be the result of getting the suggested filename[p323] for *response*.

2. Let *download behavior* be the result of WebDriver BiDi download will begin with *navigable* and a new WebDriver BiDi navigation status whose id is *navigationId*, status is "pending", url is *response*'s URL, suggestedFilename is *suggestedFilename*, and downloadResponse is *response*.

3. If *download behavior* is not null and *download behavior*'s allowed is false:

    1. Invoke WebDriver BiDi download end with *navigable* and a new WebDriver BiDi navigation status whose id is *navigationId*, status is "canceled", url is *response*'s URL, and downloadResponse is *response*.

    2. Return.

4. If *download behavior* is not null, let *destinationFolder* be *download behavior*'s destinationFolder.

5. Run these steps in parallel[p44]:

    1. Run implementation-defined steps to save *response* for later use. If *destinationFolder* is not null, the user agent should save the file to that path. If the user agent needs a filename, the user agent should use the *suggestedFilename*.

    2. If any of the following are true:

        - the download is canceled by the user;
        - the download is canceled by the user agent;
        - an error occurs (for example, a network error, not enough storage, an unavailable destination folder);

        then:

        1. Invoke WebDriver BiDi download end with *navigable* and a new WebDriver BiDi navigation status whose id is *navigationId*, status is "canceled", url is *response*'s URL.

        2. Return.

    3. When the download completes successfully, invoke WebDriver BiDi download end with *navigable* and a new WebDriver BiDi navigation status whose id is *navigationId*, status is "complete", downloadedFilepath is an absolute path of the downloaded file if available, otherwise null, url is *response*'s URL, and downloadResponse is *response*.

To **get the suggested filename** for a response *response*:

⚠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 *response* has a `Content-Disposition` header, that header specifies the attachment disposition type, and the header includes filename information, then let *filename* have the value specified by the header, and jump to the step labeled *sanitize* below. [RFC6266][p1586]

3. Let *interface origin* be the origin of the Document[p135] in which the download[p322] or navigate[p1063] action resulting in the download was initiated, if any.

4. Let *response origin* be the origin[p939] of the URL of *response*, unless that URL's scheme component is data, in which case let *response 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 *response origin* is the same origin[p940] as *interface origin*, and false otherwise.

6. If *trusted operation* is true and *response* has a `Content-Disposition` header and that header includes filename

information, then let *filename* have the value specified by the header, and jump to the step labeled *sanitize* below. [RFC6266]p1586

7. If the download was not initiated from a hyperlinkp313 created by an ap267 or areap489 element, or if the element of the hyperlinkp313 from which it was initiated did not have a downloadp314 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 filename*.

8. Let *proposed filename* have the value of the downloadp314 attribute of the element of the hyperlinkp313 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 *response* 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]p1586

11. *No proposed filename*: If *trusted operation* is true, or if the user indicated a preference for having the response in question downloaded, let *filename* have a value derived from the URL of *response* in an implementation-defined manner, and jump to the step labeled *sanitize* below.

12. Let *filename* be set to the user's preferred filename or to a filename selected by the user agent, and jump to the step labeled *sanitize* below.

    ⚠Warning!

    If the algorithm reaches this step, then a download was begun from a different origin than* response*, and the origin did not mark the file as suitable for downloading, and the download was not initiated by the user. This could be because a* downloadp314 *attribute was used to trigger the download, or because* response *is not of a type that the user agent supports.

    This could be dangerous, because, for instance, a hostile server could be trying to get a user to unknowingly download private information and then re-upload it to the hostile server, by tricking the user into thinking the data is from the hostile server.

    Thus, it is in the user's interests that the user be somehow notified that* response *comes from quite a different source, and to prevent confusion, any suggested filename from the potentially hostile* interface origin *should be ignored.

13. *Sanitize*: Optionally, allow the user to influence *filename*. For example, a user agent could prompt the user for a filename, potentially providing the value of *filename* as determined above as a default value.

14. Adjust *filename* to be suitable for the local file system.

    Example

    For example, this could involve removing characters that are not legal in filenames, or trimming leading and trailing whitespace.

15. If the platform conventions do not in any way use extensionsp325 to determine the types of file on the file system, then return *filename* as the filename.

16. Let *claimed type* be the type given by *response*'s Content-Type metadatap102, if any is known. Let *named type* be the type given by *filename*'s extensionp325, if any is known. For the purposes of this step, a *type* is a mapping of a MIME type to an extensionp325.

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 filename.

18. If *claimed type* and *named type* are the same type (i.e., the type given by *response*'s Content-Type metadatap102 is consistent with the type given by *filename*'s extensionp325), then return *filename* as the filename.

19. If the *claimed type* is known, then alter *filename* to add an extensionp325 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 extensionp325 (e.g. "`.txt`").

Note

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 filename.

For the purposes of this algorithm, a file **extension** consists of any part of the filename that platform conventions dictate will be used for identifying the type of the file. For example, many operating systems use the part of the filename following the last dot (".") in the filename to determine the type of the file, and from that the manner in which the file is to be opened or executed.

User agents should ignore any directory or path information provided by the response itself, its URL, and any download attribute, in deciding where to store the resulting file in the user's file system.

4.6.7 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 *ping URL*, ignoring when parsing returns failure:

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 *settingsObject* be the element's node document's relevant settings object.
4. Let *request* be a new request whose URL is *ping URL*, method is `POST`, header list is « (`Content-Type`, `text/ping`) », body is `PING`, client is *settingsObject*, destination is the empty string, credentials mode is "include", referrer is "no-referrer", and whose use-URL-credentials flag is set, and whose initiator type is "ping".
5. Let *target URL* be the result of encoding-parsing-and-serializing a URL given the element's href attribute's value, relative to the element's node document, and then:

   ↪ **If the URL of the Document object containing the hyperlink being audited and *ping URL* have the same origin**

   ↪ **If the origins are different, but the scheme of the URL of the Document object containing the hyperlink being audited is not "https"**

   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*. Note request *does not include a `Ping-From` header.*
6. 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.

An a or area element that creates a hyperlink and has the ping attribute is present, user agents may 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`[p314] 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`[p314] 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 UA to optimize the use of available network bandwidth so that the target page loads faster.*

4.6.7.1 The `Ping-From`[p326] and `Ping-To`[p326] headers §[p32 6]

The **`Ping-From`** and **`Ping-To`** HTTP request headers are included in hyperlink auditing[p325] requests. Their value is a URL, serialized.

4.6.8 Link types §[p32 6]

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`[p184], `a`[p267], `area`[p489], or `form`[p532] 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`[p314] 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`[p184] elements are allowed in the body[p186]. The body-ok[p326] keywords are `dns-prefetch`[p330], `modulepreload`[p335], `pingback`[p338], `preconnect`[p339], `prefetch`[p340], `preload`[p340], and `stylesheet`[p344].

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[p348].

Link type	Effect on...			body-ok[p326]	Has `Link` processing	Brief description
	`link`[p184]	`a`[p267] and `area`[p489]	`form`[p532]			
`alternate`[p327]	Hyperlink[p313]		*not allowed*	·	·	Gives alternate representations of the current document.
`canonical`[p330]	Hyperlink[p313]	*not allowed*		·	·	Gives the preferred URL for the current document.
`author`[p329]	Hyperlink[p313]		*not allowed*	·	·	Gives a link to the author of the current document or article.
`bookmark`[p329]	*not allowed*	Hyperlink[p313]	*not allowed*	·	·	Gives the permalink for the nearest ancestor section.
`dns-prefetch`[p330]	External Resource[p313]	*not allowed*		Yes	·	Specifies that the user agent should preemptively perform DNS resolution for the target resource's origin[p939].

Link type	Effect on...			body-ok	Has `Link` processing	Brief description
	link	a and area	form			
expect	Internal Resource	*not allowed*		·	·	Expect an element with the target ID to appear in the current document.
external	*not allowed*	Annotation		·	·	Indicates that the referenced document is not part of the same site as the current document.
help	Hyperlink			·	·	Provides a link to context-sensitive help.
icon	External Resource	*not allowed*		·	·	Imports an icon to represent the current document.
manifest	External Resource	*not allowed*		·	·	Imports or links to an application manifest. [MANIFEST]
modulepreload	External Resource	*not allowed*		Yes	·	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.
license	Hyperlink			·	·	Indicates that the main content of the current document is covered by the copyright license described by the referenced document.
next	Hyperlink			·	·	Indicates that the current document is a part of a series, and that the next document in the series is the referenced document.
nofollow	*not allowed*	Annotation		·	·	Indicates that the current document's original author or publisher does not endorse the referenced document.
noopener	*not allowed*	Annotation		·	·	Creates a top-level traversable with a non-auxiliary browsing context if the hyperlink would otherwise create one that was auxiliary (i.e., has an appropriate target attribute value).
noreferrer	*not allowed*	Annotation		·	·	No `Referer` (sic) header will be included. Additionally, has the same effect as noopener.
opener	*not allowed*	Annotation		·	·	Creates an auxiliary browsing context if the hyperlink would otherwise create a top-level traversable with a non-auxiliary browsing context (i.e., has "_blank" as target attribute value).
pingback	External Resource	*not allowed*		Yes	·	Gives the address of the pingback server that handles pingbacks to the current document.
preconnect	External Resource	*not allowed*		Yes	Yes	Specifies that the user agent should preemptively connect to the target resource's origin.
prefetch	External Resource	*not allowed*		Yes	·	Specifies that the user agent should preemptively fetch and cache the target resource as it is likely to be required for a followup navigation.
preload	External Resource	*not allowed*		Yes	Yes	Specifies that the user agent must preemptively fetch and cache the target resource for current navigation according to the preload destination given by the as attribute (and the priority associated with the corresponding destination).
prev	Hyperlink			·	·	Indicates that the current document is a part of a series, and that the previous document in the series is the referenced document.
privacy-policy	Hyperlink		*not allowed*	·	·	Gives a link to information about the data collection and usage practices that apply to the current document.
search	Hyperlink			·	·	Gives a link to a resource that can be used to search through the current document and its related pages.
stylesheet	External Resource	*not allowed*		Yes	·	Imports a style sheet.
tag	*not allowed*	Hyperlink	*not allowed*	·	·	Gives a tag (identified by the given address) that applies to the current document.
terms-of-service	Hyperlink		*not allowed*	·	·	Gives a link to information about the agreements between the current document's provider and users who wish to use the current document.

4.6.8.1 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[p184] elements provide some style sheets:

```
<!-- a persistent style sheet -->
<link rel="stylesheet" href="default.css">

<!-- the preferred alternate style sheet -->
<link rel="stylesheet" href="green.css" title="Green styles">

<!-- some alternate style sheets -->
<link rel="alternate stylesheet" href="contrast.css" title="High contrast">
<link rel="alternate stylesheet" href="big.css" title="Big fonts">
<link rel="alternate stylesheet" href="wide.css" title="Wide screen">
```

↪ **If the alternate[p327] keyword is used with the type[p316] attribute set to the value `application/rss+xml` or the value `application/atom+xml`**

The keyword creates a hyperlink[p313] 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[p184] elements in the document with the alternate[p327] keyword used and with their type[p316] 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[p184] elements give syndication feeds for a blog:

```
<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[p184] 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[p267] elements:

```
<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[p313] referencing an alternate representation of the current document.

The nature of the referenced document is given by the hreflang[p316], and type[p316] attributes.

If the alternate[p327] keyword is used with the hreflang[p316] attribute, and that attribute's value differs from the document element's language[p166], it indicates that the referenced document is a translation.

If the alternate[p327] keyword is used with the type[p316] attribute, it indicates that the referenced document is a reformulation of the current document in the specified format.

The hreflang[p316] and type[p316] attributes can be combined when specified with the alternate[p327] 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:

```
<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[p327]", 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.

4.6.8.2 Link type "author" §[p329]

The author[p329] keyword may be used with link[p184], a[p267], and area[p489] elements. This keyword creates a hyperlink[p313].

For a[p267] and area[p489] elements, the author[p329] keyword indicates that the referenced document provides further information about the author of the nearest article[p214] element ancestor of the element defining the hyperlink, if there is one, or of the page as a whole, otherwise.

For link[p184] elements, the author[p329] keyword indicates that the referenced document provides further information about the author for the page as a whole.

Note

The "referenced document" can be, and often is, a mailto: URL giving the email address of the author. [MAILTO][p1584]

Synonyms: For historical reasons, user agents must also treat link[p184], a[p267], and area[p489] elements that have a rev attribute with the value "made" as having the author[p329] keyword specified as a link relationship.

4.6.8.3 Link type "bookmark" §[p329]

The bookmark[p329] keyword may be used with a[p267] and area[p489] elements. This keyword creates a hyperlink[p313].

The bookmark[p329] keyword gives a permalink for the nearest ancestor article[p214] 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[p214] 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.

```
...
<body>
 <h1>Example of permalinks</h1>
 <div id="a">
  <h2>First example</h2>
  <p><a href="a.html" rel="bookmark">This permalink applies to
  only the content from the first H2 to the second H2</a>. The DIV isn't
  exactly that section, but it roughly corresponds to it.</p>
 </div>
```

```
  <h2>Second example</h2>
  <article id="b">
   <p><a href="b.html" rel="bookmark">This permalink applies to
   the outer ARTICLE element</a> (which could be, e.g., a blog post).</p>
   <article id="c">
    <p><a href="c.html" rel="bookmark">This permalink applies to
    the inner ARTICLE element</a> (which could be, e.g., a blog comment).</p>
   </article>
  </article>
 </body>
 ...
```

4.6.8.4 Link type "canonical" §[p33]

The canonical[p330] keyword may be used with link[p184] element. This keyword creates a hyperlink[p313].

The canonical[p330] keyword indicates that URL given by the href[p185] 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][p1586]

MDN

4.6.8.5 Link type "dns-prefetch" §[p33]

The dns-prefetch[p330] keyword may be used with link[p184] elements. This keyword creates an external resource link[p313]. This keyword is body-ok[p326].

The dns-prefetch[p330] keyword indicates that preemptively performing DNS resolution for the origin[p939] of the specified resource is likely to be beneficial, as it is highly likely that the user will require resources located at that origin[p939], and the user experience would be improved by preempting the latency costs associated with DNS resolution.

There is no default type for resources given by the dns-prefetch[p330] keyword.

The appropriate times to fetch and process[p190] this type of link are:

- When the external resource link[p313] is created on a link[p184] element that is already browsing-context connected[p47].
- When the external resource link[p313]'s link[p184] element becomes browsing-context connected[p47].
- When the href[p185] attribute of the link[p184] element of an external resource link[p313] that is already browsing-context connected[p47] is changed.

The fetch and process the linked resource[p190] steps for this type of linked resource, given a link[p184] element *el*, are:

1. Let *url* be the result of encoding-parsing a URL[p101] given *el*'s href[p185] attribute's value, relative to *el*'s node document.
2. If *url* is failure, then return.
3. Let *partitionKey* be the result of determining the network partition key given *el*'s node document's relevant settings object[p1151].
4. The user agent should resolve an origin given *partitionKey* and *url*'s origin.

Note

As the results of this algorithm can be cached, future fetches could be faster.

4.6.8.6 Link type "expect" §[p33]

The expect[p330] keyword may be used with link[p184] elements. This keyword creates an internal resource link[p313].

An internal resource link[p313] created by the `expect`[p330] keyword can be used to block rendering[p142] until the element that it indicates[p1104] is connected to the document and fully parsed.

There is no default type for resources given by the `expect`[p330] keyword.

Whenever any of the following conditions occur for a `link`[p184] element *el*:

- the `expect`[p330] internal resource link[p313] is created on *el* that is already browsing-context connected[p47];
- an `expect`[p330] internal resource link[p313] has been created on *el* and *el* becomes browsing-context connected[p47];
- an `expect`[p330] internal resource link[p313] has been created on *el*, *el* is already browsing-context connected[p47], and *el*'s `href`[p185] attribute is set, changed, or removed; or
- an `expect`[p330] internal resource link[p313] has been created on *el*, *el* is already browsing-context connected[p47], and *el*'s `media`[p186] attribute is set, changed, or removed,

then process[p331] *el*.

To **process internal resource link** given a `link`[p184] element *el*, run these steps:

1. Let *doc* be *el*'s node document.
2. Let *url* be the result of encoding-parsing a URL[p101] given *el*'s `href`[p185] attribute's value, relative to *doc*.
3. If this fails, or if *url* does not equal *doc*'s URL with *exclude fragments* set to true, then unblock rendering[p142] on *el* and return.
4. Let *indicatedElement* be the result of selecting the indicated part[p1104] given *doc* and *url*.
5. If all of the following are true:
    - *doc*'s current document readiness[p141] is "`loading`";
    - *el* creates an internal resource link[p313];
    - *el* is browsing-context connected[p47];
    - *el*'s `rel`[p185] attribute contains `expect`[p330];
    - *el* is potentially render-blocking[p107];
    - *el*'s `media`[p186] attribute matches the environment[p100]; and
    - *indicatedElement* is not an element, or is on a stack of open elements[p1383] of an HTML parser[p1368] whose associated `Document`[p135] is *doc*,

    then block rendering[p142] on *el*.
6. Otherwise, unblock rendering[p142] on *el*.

To **process internal resource links** given a `Document`[p135] *doc*:

1. For each `expect`[p330] `link`[p184] element *link* in *doc*'s render-blocking element set[p142], process[p331] *link*.

The following attribute change steps, given *element*, *localName*, *oldValue*, *value*, and *namespace*, are used to ensure `expect`[p330] `link`[p184] elements respond to dynamic `id`[p162] and `name`[p1531] changes:

1. If *namespace* is not null, then return.
2. If *element* is in a stack of open elements[p1383] of an HTML parser[p1368], then return.
3. If any of the following is true:
    - *localName* is `id`[p162]; or
    - *localName* is `name`[p1531] and *element* is an `a`[p267] element,

    then process internal resource links[p331] given *element*'s node document.

4.6.8.7 Link type "external" §

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.

4.6.8.8 Link type "help" §

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.

```
<p><label> Topic: <input name=topic> <a href="help/topic.html" rel="help">(Help)</a></label></p>
```

For link elements, the help keyword indicates that the referenced document provides help for the page as a whole.

For a and area elements, on some browsers, the help keyword causes the link to use a different cursor.

4.6.8.9 Link type "icon" §

The icon keyword may be used with link elements. This keyword creates an external resource link.

The specified resource is an icon representing the page or site, and should be used by the user agent when representing the page in the user interface.

Icons could be auditory icons, visual icons, or other kinds of icons. If multiple icons are provided, the user agent must select the most appropriate icon according to the type, media, and sizes attributes. If there are multiple equally appropriate icons, user agents must use the last one declared in tree order at the time that the user agent collected the list of icons. If the user agent tries to use an icon but that icon is determined, upon closer examination, to in fact be inappropriate (e.g. because it uses an unsupported format), then the user agent must try the next-most-appropriate icon as determined by the attributes.

Note

User agents are not required to update icons when the list of icons changes, but are encouraged to do so.

There is no default type for resources given by the icon keyword. However, for the purposes of determining the type of the resource, user agents must expect the resource to be an image.

The sizes keywords represent icon sizes in raw pixels (as opposed to CSS pixels).

Note

An icon that is 50 CSS pixels wide intended for displays with a device pixel density of two device pixels per CSS pixel (2x, 192dpi) would have a width of 100 raw pixels. This feature does not support indicating that a different resource is to be used for small high-resolution icons vs large low-resolution icons (e.g. 50×50 2x vs 100×100 1x).

To parse and process the attribute's value, the user agent must first split the attribute's value on ASCII whitespace, and must then parse each resulting keyword to determine what it represents.

The **any** keyword represents that the resource contains a scalable icon, e.g. as provided by an SVG image.

Other keywords must be further parsed as follows to determine what they represent:

1. 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.
2. Let *width string* be the string before the "x" or "X".
3. Let *height string* be the string after the "x" or "X".
4. If either *width string* or *height string* start with a U+0030 DIGIT ZERO (0) character or contain any characters other than ASCII digits, then this keyword doesn't represent anything. Return for that keyword.
5. Apply the rules for parsing non-negative integers[p81] to *width string* to obtain *width*.
6. Apply the rules for parsing non-negative integers[p81] to *height string* to obtain *height*.
7. The keyword represents that the resource contains a bitmap icon with a width of *width* device pixels and a height of *height* device pixels.

The keywords specified on the `sizes`[p188] attribute must not represent icon sizes that are not actually available in the linked resource.

The linked resource fetch setup steps[p190] for this type of linked resource, given a `link`[p184] element *el* and request *request*, are:

1. Set *request*'s destination to "image".
2. Return true.

The process a link header[p191] steps for this type of linked resource are to do nothing.

In the absence of a `link`[p184] with the `icon`[p332] keyword, for `Document`[p135] objects whose URL's scheme is an HTTP(S) scheme, user agents may instead run these steps:

1. Let *request* be a new request whose URL is the URL record obtained by resolving the URL "/favicon.ico" against the `Document`[p135] object's URL, client is the `Document`[p135] object's relevant settings object[p1151], destination is "image", credentials mode is "include", and whose use-URL-credentials flag is set.
2. Fetch *request*, with *processResponse* set to the following steps given response *response*:
    1. Use *response*'s unsafe response[p102] as an icon as if it had been declared using the `icon`[p332] keyword.

Example

The following snippet shows the top part of an application with several icons.

```
<!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>
  <meta name=application-name content="lsForums">
 </head>
 <body>
  ...
```

For historical reasons, the `icon`[p332] keyword may be preceded by the keyword "shortcut". If the "shortcut" keyword is present, the `rel`[p314] 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).

4.6.8.10 Link type "license" §[p334]

The license[p334] keyword may be used with link[p184], a[p267], area[p489], and form[p532] elements. This keyword creates a hyperlink[p313].

The license[p334] 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:

```
<!DOCTYPE HTML>
<html lang="en">
 <head>
  <title>Exampl 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><a rel="license" href="http://www.opensource.org/licenses/mit-license.php">MIT
Licensed</a></small></p>
  <footer>
   <a href="/">Home</a> | <a href="../">Photo index</a>
   <p><small>© copyright 2009 Exampl Pictures. All Rights Reserved.</small></p>
  </footer>
 </body>
</html>
```

In this case the license[p334] 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[p334] keyword.

4.6.8.11 Link type "manifest" §[p334]

The manifest[p334] keyword may be used with link[p184] elements. This keyword creates an external resource link[p313].

The manifest[p334] keyword indicates the manifest file that provides metadata associated with the current document.

There is no default type for resources given by the manifest[p334] keyword.

When a web application is not installed, the appropriate time to fetch and process the linked resource[p190] for this link type is when the user agent deems it necessary. For example, when the user chooses to install the web application.

For an installed web application, the appropriate times to fetch and process the linked resource[p190] for this link type are:

- When the external resource link[p313] is created on a link[p184] element that is already browsing-context connected[p47].

- When the external resource link[p313]'s `link`[p184] element becomes browsing-context connected[p47].
- When the `href`[p185] attribute of the `link`[p184] element of an external resource link[p313] that is already browsing-context connected[p47] is changed.

In any case, only the first `link`[p184] element in tree order whose `rel`[p185] attribute contains the token `manifest`[p334] may be used.

A user agent must not delay the load event[p1458] for this link type.

The linked resource fetch setup steps[p190] for this type of linked resource, given a `link`[p184] element *el* and request *request*, are:

1. Let *navigable* be *el*'s node document's node navigable[p1036].
2. If *navigable* is null, then return false.
3. If *navigable* is not a top-level traversable[p1037], 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[p104] for *el*'s `crossorigin`[p186] content attribute.
8. Return true.

To process this type of linked resource[p191] given a `link`[p184] element *el*, boolean *success*, response *response*, and byte sequence *bodyBytes*:

1. If *response*'s Content-Type metadata[p102] is not a JSON MIME type, then set *success* to false.
2. If *success* is true:
    1. Let *document URL* be *el*'s node document's URL.
    2. Let *manifest URL* be *response*'s URL.
    3. Let *client* be *el*'s node document's relevant settings object[p1151].
    4. Process the manifest given *document URL*, *manifest URL*, *bodyBytes*, and *client*. [MANIFEST][p1584]

The process a link header[p191] steps for this type of linked resource are to do nothing.

MDN

4.6.8.12 Link type "`modulepreload`" §[p335]

The `modulepreload`[p335] keyword may be used with `link`[p184] elements. This keyword creates an external resource link[p313]. This keyword is body-ok[p326].

The `modulepreload`[p335] keyword is a specialized alternative to the `preload`[p340] keyword, with a processing model geared toward preloading module scripts[p1153]. In particular, it uses the specific fetch behavior for module scripts (including, e.g., a different interpretation of the `crossorigin`[p186] attribute), and places the result into the appropriate module map[p136] for later evaluation. In contrast, a similar external resource link[p313] using the `preload`[p340] keyword would place the result in the preload cache, without affecting the document's module map[p136].

Additionally, implementations can take advantage of the fact that module scripts[p1153] 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`[p1575] or `error`[p1575] events will occur after the specified module is fetched, and will not wait for any dependencies.

A user agent must not delay the load event[p1458] for this link type.

A **module preload destination** is "`json`", "`style`", "`text`", or a script-like destination.

The appropriate times to fetch and process the linked resource[p190] for such a link are:

- When the external resource link[p313] is created on a link[p184] element that is already browsing-context connected[p47].
- When the external resource link[p313]'s link[p184] element becomes browsing-context connected[p47].
- When the href[p185] attribute of the link[p184] element of an external resource link[p313] that is already browsing-context connected[p47] is changed.

Note

Unlike some other link relations, changing the relevant attributes (such as as[p188], crossorigin[p186], and referrerpolicy[p187]) of such a link[p184] does not trigger a new fetch. This is because the document's module map[p136] has already been populated by a previous fetch, and so re-fetching would be pointless.

The fetch and process the linked resource[p190] algorithm for modulepreload[p335] links, given a link[p184] element *el*, is as follows:

1. If *el*'s href[p185] attribute's value is the empty string, then return.
2. Let *destination* be the current state of *el*'s as[p188] attribute (a destination), or "`script`" if it is in no state.
3. If *destination* is not a module preload destination[p335], then queue an element task[p1195] on the networking task source[p1204] given *el* to fire an event named error[p1575] at *el*, and return.
4. Let *url* be the result of encoding-parsing a URL[p101] given *el*'s href[p185] attribute's value, relative to *el*'s node document.
5. If *url* is failure, then return.
6. Let *settings object* be *el*'s node document's relevant settings object[p1151].
7. Let *credentials mode* be the CORS settings attribute credentials mode[p104] for *el*'s crossorigin[p186] attribute.
8. Let *cryptographic nonce* be *el*.[[CryptographicNonce]][p104].
9. Let *integrity metadata* be the value of *el*'s integrity[p186] attribute, if it is specified, or the empty string otherwise.
10. If *el* does not have an integrity[p186] attribute, then set *integrity metadata* to the result of resolving a module integrity metadata[p1155] with *url* and *settings object*.
11. Let *referrer policy* be the current state of *el*'s referrerpolicy[p187] attribute.
12. Let *fetch priority* be the current state of *el*'s fetchpriority[p189] attribute.
13. Let *options* be a script fetch options[p1154] whose cryptographic nonce[p1154] is *cryptographic nonce*, integrity metadata[p1154] is *integrity metadata*, parser metadata[p1154] is "`not-parser-inserted`", credentials mode[p1154] is *credentials mode*, referrer policy[p1154] is *referrer policy*, and fetch priority[p1154] is *fetch priority*.
14. Fetch a modulepreload module script graph[p1158] given *url*, *destination*, *settings object*, *options*, and with the following steps given *result*:
    1. If *result* is null, then fire an event named error[p1575] at *el*, and return.
    2. Fire an event named load[p1575] at *el*.

The process a link header[p191] steps for this type of linked resource are to do nothing.

¶[p336] **Example**

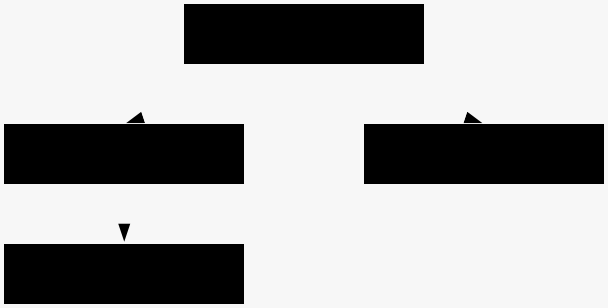
The following snippet shows the top part of an application with several modules preloaded:

```
<!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:



Here we see the application developer has used `modulepreload`[p335] 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`[p335] `link`[p184] elements can be used as a sort of "manifest" of the application's modules.

¶[p337] Example

The following code shows how `modulepreload`[p335] 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[p1189]:

```
<link rel="modulepreload" href="awesome-viewer.mjs">

<button onclick="import('./awesome-viewer.mjs').then(m => m.view())">
  View awesome thing
</button>
```

4.6.8.13 Link type "nofollow" §[p337]

The `nofollow`[p337] keyword may be used with `a`[p267], `area`[p489], and `form`[p532] elements. This keyword does not create a hyperlink[p313], but annotates[p314] any other hyperlinks created by the element (the implied hyperlink, if no other keywords create one).

The `nofollow`[p337] 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.

4.6.8.14 Link type "noopener" §[p337]

The `noopener`[p337] keyword may be used with `a`[p267], `area`[p489], and `form`[p532] elements. This keyword does not create a hyperlink[p313], but annotates[p314] 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 traversable[p1037] which results from following the hyperlink[p313] will not contain an auxiliary browsing context[p1046]. E.g., the resulting `Window`[p965]'s `opener`[p973] getter will return null.

Note

See also the processing model[p1045].

Example

This typically creates a top-level traversable[p1037] with an auxiliary browsing context[p1046] (assuming there is no existing navigable[p1035] whose target name[p1036] is "`example`"):

```
<a href=help.html target=example>Help!</a>
```

This creates a top-level traversable[p1037] with a non-auxiliary browsing context[p1046] (assuming the same thing):

```
<a href=help.html target=example rel=noopener>Help!</a>
```

These are equivalent and only navigate the parent navigable[p1035]:

```
<a href=index.html target=_parent>Home</a>
```

```
<a href=index.html target=_parent rel=noopener>Home</a>
```

4.6.8.15 Link type "noreferrer" §[p338]

The noreferrer[p338] keyword may be used with a[p267], area[p489], and form[p532] elements. This keyword does not create a hyperlink[p313], but annotates[p314] 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 the noopener[p337] keyword behavior under the same conditions.

Note

See also the processing model[p322] where referrer is directly manipulated.

Example

`<a href="..." rel="noreferrer" target="_blank">` has the same behavior as `<a href="..." rel="noreferrer noopener" target="_blank">`.

4.6.8.16 Link type "opener" §[p338]

The opener[p338] keyword may be used with a[p267], area[p489], and form[p532] elements. This keyword does not create a hyperlink[p313], but annotates[p314] 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 traversable[p1037] which results from following the hyperlink[p313] will contain an auxiliary browsing context[p1046].

Note

See also the processing model[p321].

Example

In the following example the opener[p338] 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.

```
<a href="..." rel=opener target=_blank>Help!</a>
```

4.6.8.17 Link type "pingback" §[p338]

The pingback[p338] keyword may be used with link[p184] elements. This keyword creates an external resource link[p313]. This keyword is body-ok[p326].

For the semantics of the pingback[p338] keyword, see *Pingback 1.0*. [PINGBACK][p1585]

4.6.8.18 Link type "preconnect" §[p339]

The preconnect[p339] keyword may be used with link[p184] elements. This keyword creates an external resource link[p313]. This keyword is body-ok[p326].

The preconnect[p339] keyword indicates that preemptively initiating a connection to the origin[p939] of the specified resource is likely to be beneficial, as it is highly likely that the user will require resources located at that origin[p939], and the user experience would be improved by preempting the latency costs associated with establishing the connection.

There is no default type for resources given by the preconnect[p339] keyword.

A user agent must not delay the load event[p1458] for this link type.

The appropriate times to fetch and process[p190] this type of link are:

- When the external resource link[p313] is created on a link[p184] element that is already browsing-context connected[p47].
- When the external resource link[p313]'s link[p184] element becomes browsing-context connected[p47].
- When the href[p185] attribute of the link[p184] element of an external resource link[p313] that is already browsing-context connected[p47] is changed.
- When the crossorigin[p186] attribute of the link[p184] element of an external resource link[p313] that is already browsing-context connected[p47] is set, changed, or removed.

The fetch and process the linked resource[p190] steps for this type of linked resource, given a link[p184] element *el*, are to create link options[p192] from *el* and to preconnect[p339] given the result.

The process a link header[p191] step for this type of linked resource given a link processing options[p191] *options* are to preconnect[p339] given *options*.

To **preconnect** given a link processing options[p191] *options*:

1. If *options*'s href[p192] is an empty string, return.
2. Let *url* be the result of encoding-parsing a URL[p101] given *options*'s href[p192], relative to *options*'s base URL[p192].

   Passing the base URL instead of a document or environment is tracked by issue #9715.
3. If *url* is failure, then return.
4. If *url*'s scheme is not an HTTP(S) scheme, then return.
5. Let *partitionKey* be the result of determining the network partition key given *options*'s environment[p192].
6. Let *useCredentials* be true.
7. If *options*'s crossorigin[p192] is Anonymous[p103] and *options*'s origin[p192] does not have the same origin[p940] as *url*'s origin, then set *useCredentials* to false.
8. The user agent should obtain a connection given *partitionKey*, *url*'s origin, and *useCredentials*.

   Note

   This connection is obtained but not used directly. It will remain in the connection pool for subsequent use.

   The user agent should attempt to initiate a preconnect and perform the full connection handshake (DNS+TCP for HTTP, and DNS+TCP+TLS for HTTPS origins) whenever possible, but is allowed to elect to perform a partial handshake (DNS only for HTTP, and DNS or DNS+TCP for HTTPS origins), or skip it entirely, due to resource constraints or other reasons.

   The optimal number of connections per origin is dependent on the negotiated protocol, users current connectivity profile, available device resources, global connection limits, and other context specific variables. As a result, the decision for how many connections should be opened is deferred to the user agent.

4.6.8.19 Link type "prefetch" §p34

The prefetchp340 keyword may be used with linkp184 elements. This keyword creates an external resource linkp313. This keyword is body-okp326.

The prefetchp340 keyword indicates that preemptively fetching and caching the specified resource or same-site document is likely to be beneficial, as it is highly likely that the user will require this resource for future navigations.

There is no default type for resources given by the prefetchp340 keyword.

The appropriate times to fetch and processp190 this type of link are:

- When the external resource linkp313 is created on a linkp184 element that is already browsing-context connectedp47.
- When the external resource linkp313's linkp184 element becomes browsing-context connectedp47.
- When the hrefp185 attribute of the linkp184 element of an external resource linkp313 that is already browsing-context connectedp47 is changed.
- When the crossoriginp186 attribute of the linkp184 element of an external resource linkp313 that is already browsing-context connectedp47 is set, changed, or removed.

The fetch and process the linked resourcep190 algorithm for prefetchp340 links, given a linkp184 element *el*, is as follows:

1. If *el*'s hrefp185 attribute's value is the empty string, then return.
2. Let *options* be the result of creating link optionsp192 from *el*.
3. Let *request* be the result of creating a link requestp191 given *options*.
4. If *request* is null, then return.
5. Set *request*'s initiator to "prefetch".
6. Let *processPrefetchResponse* be the following steps given a response *response* and null, failure, or a byte sequence *bytesOrNull*:
    1. If *response* is a network error, fire an event named errorp1575 at *el*.
    2. Otherwise, fire an event named loadp1575 at *el*.
7. The user agent should fetch *request*, with *processResponseConsumeBody* set to *processPrefetchResponse*. User agents may delay the fetching of *request* to prioritize other requests that are necessary for the current document.

The process a link headerp191 steps for this type of linked resource are to do nothing.

4.6.8.20 Link type "preload" §p34

The preloadp340 keyword may be used with linkp184 elements. This keyword creates an external resource linkp313. This keyword is body-okp326.

The preloadp340 keyword indicates that the user agent will preemptively fetch and cache the specified resource according to the preload destinationp342 given by the asp188 attribute, and the priority given by the fetchpriorityp189 attribute, as it is highly likely that the user will require this resource for the current navigation.

Note

User-agents might perform additional operations when a resource is loaded, such as preemptively decoding imagesp365 or creating style sheets. However, these additional operations cannot have observable effects.

There is no default type for resources given by the preloadp340 keyword.

A user agent must not delay the load eventp1458 for this link type.

The appropriate times to fetch and process the linked resourcep190 for such a link are:

- When the external resource link[p313] is created on a `link`[p184] element that is already browsing-context connected[p47].
- When the external resource link[p313]'s `link`[p184] element becomes browsing-context connected[p47].
- When the `href`[p185] attribute of the `link`[p184] element of an external resource link[p313] that is already browsing-context connected[p47] is changed.
- When the `as`[p188] attribute of the `link`[p184] element of an external resource link[p313] that is already browsing-context connected[p47] is changed.
- When the `type`[p186] attribute of the `link`[p184] element of an external resource link[p313] that is already browsing-context connected[p47], but was previously not obtained due to the `type`[p186] attribute specifying an unsupported type for the request destination, is set, removed, or changed.
- When the `media`[p186] attribute of the `link`[p184] element of an external resource link[p313] that is already browsing-context connected[p47], but was previously not obtained due to the `media`[p186] attribute not matching the environment[p100], is changed or removed.

A `Document`[p135] has a **map of preloaded resources**, which is an ordered map, initially empty.

A **preload key** is a struct. It has the following items:

URL
A URL

destination
A preload destination[p342]

mode
A request mode, either "`same-origin`", "`cors`", or "`no-cors`"

credentials mode
A credentials mode

A preload key[p341]'s equality is itemwise equality.

A **preload entry** is a struct. It has the following items:

integrity metadata
A string

response
Null or a response

on response available
Null, or an algorithm accepting a response or null

To **consume a preloaded resource** for `Window`[p965] *window*, given a URL *url*, a string *destination*, a string *mode*, a string *credentialsMode*, a string *integrityMetadata*, and *onResponseAvailable*, which is an algorithm accepting a response:

1. Let *key* be a preload key[p341] whose URL[p341] is *url*, destination[p342] is *destination*, mode[p341] is *mode*, and credentials mode[p341] is *credentialsMode*.
2. Let *preloads* be *window*'s associated `Document`[p966]'s map of preloaded resources[p341].
3. If *key* does not exist in *preloads*, then return false.
4. Let *entry* be *preloads*[*key*].
5. Let *consumerIntegrityMetadata* be the result of parsing *integrityMetadata*.
6. Let *preloadIntegrityMetadata* be the result of parsing *entry*'s integrity metadata[p341].
7. If none of the following conditions apply:
    - *consumerIntegrityMetadata* is `no metadata`;
    - *consumerIntegrityMetadata* is equal to *preloadIntegrityMetadata*; or

This comparison would ignore unknown integrity options. See issue #116.

then return false.

Note

A mismatch in integrity metadata between the preload and the consumer, even if both match the data, would lead to an additional fetch from the network.

Note

It is important that network errors are added to the preload cache so that if a preload request results in an error, the erroneous response isn't re-requested from the network later. This also has security implications; consider the case where a developer specifies subresource integrity metadata on a preload request, but not the following resource request. If the preload request fails subresource integrity verification and is discarded, the resource request will fetch and consume a potentially-malicious response from the network without verifying its integrity. [SRI][p1586]

8. Remove *preloads*[*key*].
9. If *entry*'s response[p341] is null, then set *entry*'s on response available[p341] to *onResponseAvailable*.
10. Otherwise, call *onResponseAvailable* with *entry*'s response[p341].
11. Return true.

For the purposes of this section, a string *type* **matches** a preload destination[p342] *destination* if the following algorithm returns true:

1. If *type* is an empty string, then return true.
2. If *destination* is "`fetch`", then return true.
3. Let *mimeTypeRecord* be the result of parsing *type*.
4. If *mimeTypeRecord* is failure, then return false.
5. If *mimeTypeRecord* is not supported by the user agent, then return false.
6. If any of the following are true:
    - *destination* is "`script`" and *mimeTypeRecord* is a JavaScript MIME type;
    - *destination* is "`image`" and *mimeTypeRecord* is an image MIME type;
    - *destination* is "`font`" and *mimeTypeRecord* is a font MIME type;
    - *destination* is "`style`" and *mimeTypeRecord*'s essence is `text/css`[p1577]; or
    - *destination* is "`track`" and *mimeTypeRecord*'s essence is `text/vtt`[p1578],

    then return true.
7. Return false.

To **create a preload key** for a request *request*, return a new preload key[p341] whose URL[p341] is *request*'s URL, destination[p342] is *request*'s destination, mode[p341] is *request*'s mode, and credentials mode[p341] is *request*'s credentials mode.

A **preload destination** is "`fetch`", "`font`", "`image`", "`script`", "`style`", or "`track`".

To **translate a preload destination** given a string *destination*:

1. If *destination* is not a preload destination[p342], then return null.
2. Return the result of translating *destination*.

To **preload** given a link processing options[p191] *options* and an optional *processResponse*, which is an algorithm accepting a response:

1. If *options*'s type[p192] doesn't match[p342] *options*'s destination[p192], then return.
2. If *options*'s destination[p192] is "`image`" and *options*'s source set[p192] is not null, then set *options*'s href[p192] to the result of selecting an image source[p385] from *options*'s source set[p192].

3. Let *request* be the result of creating a link request[p191] given *options*.
4. If *request* is null, then return.
5. Let *unsafeEndTime* be 0.
6. Let *entry* be a new preload entry[p341] whose integrity metadata[p341] is *options*'s integrity[p192].
7. Let *key* be the result of creating a preload key[p342] given *request*.
8. If *options*'s document[p192] is null, then set *request*'s initiator type to "`early hint`".
9. Let *controller* be null.
10. Let *reportTiming* given a `Document`[p135] *document* be to report timing for *controller* given *document*'s relevant global object[p1151].
11. Set *controller* to the result of fetching *request*, with *processResponseConsumeBody* set to the following steps given a response *response* and null, failure, or a byte sequence *bodyBytes*:
    1. If *bodyBytes* is a byte sequence, then set *response*'s body to *bodyBytes* as a body.

       Note

       By using processResponseConsumeBody, we have extracted the entire body. This is necessary to ensure the preloader loads the entire body from the network, regardless of whether the preload will be consumed (which is uncertain at this point). This step then resets the request's body to a new body containing the same bytes, so that other specifications can read from it at the time of actual consumption, despite us having already done so once.
    2. Otherwise, set *response* to a network error.
    3. Set *unsafeEndTime* to the unsafe shared current time.
    4. If *options*'s document[p192] is not null, then call *reportTiming* given *options*'s document[p192].
    5. If *entry*'s on response available[p341] is null, then set *entry*'s response[p341] to *response*; otherwise call *entry*'s on response available[p341] given *response*.
    6. If *processResponse* is given, then call *processResponse* with *response*.
12. Let *commit* be the following steps given a `Document`[p135] *document*:
    1. If *entry*'s response[p341] is not null, then call *reportTiming* given *document*.
    2. Set *document*'s map of preloaded resources[p341][*key*] to *entry*.
13. If *options*'s document[p192] is null, then set *options*'s on document ready[p192] to *commit*. Otherwise, call *commit* with *options*'s document[p192].

The fetch and process the linked resource[p190] steps for this type of linked resource, given a `link`[p184] element *el*, are:

1. Update the source set[p386] for *el*.
2. Let *options* be the result of creating link options[p192] from *el*.
3. Let *destination* be the result of translating[p342] the keyword representing the state of *el*'s `as`[p188] attribute.
4. If *destination* is null, then return.
5. Set *options*'s destination[p192] to *destination*.
6. Preload[p342] *options*, with the following steps given a response *response*:
    1. If *response* is a network error, fire an event named `error`[p1575] at *el*. Otherwise, fire an event named `load`[p1575] at *el*.

       The actual browsers' behavior is different from the spec here, and the feasibility of changing the behavior has not yet been investigated. See issue #1142.

The process a link header[p191] step for this type of link given a link processing options[p191] *options* is to preload[p342] *options*.

4.6.8.21 Link type "privacy-policy" §p344 4

The privacy-policy[p344] keyword may be used with link[p184], a[p267], and area[p489] elements. This keyword creates a hyperlink[p313].

The privacy-policy[p344] keyword indicates that the referenced document contains information about the data collection and usage practices that apply to the current document, as described in more detail in *Additional Link Relation Types*. The referenced document may be a standalone privacy policy, or a specific section of some more general document. [RFC6903][p1586]

4.6.8.22 Link type "search" §p344 4

The search[p344] keyword may be used with link[p184], a[p267], area[p489], and form[p532] elements. This keyword creates a hyperlink[p313].

The search[p344] keyword indicates that the referenced document provides an interface specifically for searching the document and its related resources.

Note

OpenSearch description documents can be used with link[p184] elements and the search[p344] link type to enable user agents to autodiscover search interfaces. [OPENSEARCH][p1584]

4.6.8.23 Link type "stylesheet" §p344 4

The stylesheet[p344] keyword may be used with link[p184] elements. This keyword creates an external resource link[p313] that contributes to the styling processing model. This keyword is body-ok[p326].

The specified resource is a CSS style sheet that describes how to present the document.

If the alternate[p327] keyword is also specified on the link[p184] element, then **the link is an alternative style sheet**; in this case, the title[p164] attribute must be specified on the link[p184] element, with a non-empty value.

The default type for resources given by the stylesheet[p344] keyword is text/css[p1577].

A link[p184] element of this type is implicitly potentially render-blocking[p107] if the element was created by its node document's parser.

When the disabled[p188] attribute of a link[p184] element with a stylesheet[p344] keyword is set, disable the associated CSS style sheet.

The appropriate times to fetch and process[p190] this type of link are:

- When the external resource link[p313] is created on a link[p184] element that is already browsing-context connected[p47].
- When the external resource link[p313]'s link[p184] element becomes browsing-context connected[p47].
- When the href[p185] attribute of the link[p184] element of an external resource link[p313] that is already browsing-context connected[p47] is changed.
- When the disabled[p188] attribute of the link[p184] element of an external resource link[p313] that is already browsing-context connected[p47] is set, changed, or removed.
- When the crossorigin[p186] attribute of the link[p184] element of an external resource link[p313] that is already browsing-context connected[p47] is set, changed, or removed.
- When the type[p186] attribute of the link[p184] element of an external resource link[p313] that is already browsing-context connected[p47] is set or changed to a value that does not or no longer matches the Content-Type metadata[p102] of the previous obtained external resource, if any.
- When the type[p186] attribute of the link[p184] element of an external resource link[p313] that is already browsing-context connected[p47], but was previously not obtained due to the type[p186] attribute specifying an unsupported type, is removed or changed.
- When the external resource link[p313] that is already browsing-context connected[p47] changes from being an alternative style sheet[p344] to not being one, or vice versa.

Quirk: If the document has been set to quirks mode, has the same origin[p940] as the URL of the external resource, and the Content-Type

metadata[p102] of the external resource is not a supported style sheet type, the user agent must instead assume it to be `text/css`[p1577].

The linked resource fetch setup steps[p190] for this type of linked resource, given a `link`[p184] element *el* and request *request*, are:

1. If *el*'s `disabled`[p188] attribute is set, then return false.
2. If *el* contributes a script-blocking style sheet[p211], append *el* to its node document's script-blocking style sheet set[p212].
3. If *el*'s `media`[p186] attribute's value matches the environment[p100] and *el* is potentially render-blocking[p107], then block rendering[p142] on *el*.
4. If *el* is currently render-blocking[p142], then set *request*'s render-blocking to true.
5. 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[p190] algorithm. In the meantime, any critical subresource[p46] request should have its render-blocking set to whether or not the `link`[p184] element is currently render-blocking[p142].

To process this type of linked resource[p191] given a `link`[p184] element *el*, boolean *success*, response *response*, and byte sequence *bodyBytes*:

1. If the resource's Content-Type metadata[p102] is not `text/css`[p1577], then set *success* to false.
2. If *el* no longer creates an external resource link[p313] that contributes to the styling processing model, or if, since the resource in question was fetched[p190], it has become appropriate to fetch[p190] it again:
    1. Remove *el* from *el*'s node document's script-blocking style sheet set[p212].
    2. Return.
3. If *el* has an associated CSS style sheet, remove the CSS style sheet.
4. If *success* is true:
    1. Create a CSS style sheet with the following properties:

        type

        `text/css`[p1577]

        location

        response's URL list[0]

        We provide a URL here on the assumption that w3c/csswg-drafts issue #9316 will be fixed.

        owner node

        el

        media

        The `media`[p186] attribute of *el*.

        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`[p187] attribute of *el*, if *el* 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[p344] and *el*'s explicitly enabled[p188] is false; unset otherwise.

origin-clean flag

Set if the resource is CORS-same-origin[p102]; 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 *bodyBytes*? Tracked as issue #2997.

The CSS environment encoding is the result of running the following steps: [CSSSYNTAX][p1581]

1. If *el* has a `charset`[p1531] attribute, get an encoding from that attribute's value. If that succeeds, return the resulting encoding. [ENCODING][p1582]
2. Otherwise, return the document's character encoding. [DOM][p1582]

2. Fire an event named `load`[p1575] at *el*.

5. Otherwise, fire an event named `error`[p1575] at *el*.
6. If *el* contributes a script-blocking style sheet[p211]:
    1. Assert: *el*'s node document's script-blocking style sheet set[p212] contains *el*.
    2. Remove *el* from its node document's script-blocking style sheet set[p212].
7. Unblock rendering[p142] on *el*.

The process a link header[p191] steps for this type of linked resource are to do nothing.

4.6.8.24 Link type "tag" §[p346]

The `tag`[p346] keyword may be used with `a`[p267] and `area`[p489] elements. This keyword creates a hyperlink[p313].

The `tag`[p346] 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[p814], 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:

```
<!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[p346]" link, however, applies to the whole page (and would do so wherever it was placed, including if it was within the article[p214] elements).

```
<!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/
Trains_de_la_Bernina_en_hiver_2.jpg"
         alt="The 802 was red with pantographs and tall vents on the side.">
    <figcaption>The 802 in the 1980s, above Lago Bianco.</figcaption>
   </figure>
   <p>The number 802 Gem 4/4 electro-diesel has a marmot and was rebuilt in 2003.</p>
  </article>
  <p class="topic"><a rel=tag href="https://en.wikipedia.org/wiki/Rhaetian_Railway_Gem_4/4">Gem
4/4</a></p>
 </body>
</html>
```

4.6.8.25 Link Type "terms-of-service" §[p347]

The terms-of-service[p347] keyword may be used with link[p184], a[p267], and area[p489] elements. This keyword creates a hyperlink[p313].

The terms-of-service[p347] keyword indicates that the referenced document contains information about the agreements between the current document's provider and users who wish to use the current document, as described in more detail in *Additional Link Relation Types*. [RFC6903][p1586]

4.6.8.26 Sequential link types §[p347]

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.

4.6.8.26.1 Link type "next" §p34

The next[p348] keyword may be used with link[p184], a[p267], area[p489], and form[p532] elements. This keyword creates a hyperlink[p313].

The next[p348] 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[p348] keyword is used with a link[p184] element, user agents should process such links as if they were using one of the dns-prefetch[p330], preconnect[p339], or prefetch[p340] keywords. Which keyword the user agent wishes to use is implementation-dependent; for example, a user agent may wish to use the less-costly preconnect[p339] processing model when trying to conserve data, battery power, or processing power, or may wish to pick a keyword depending on heuristic analysis of past user behavior in similar scenarios.

4.6.8.26.2 Link type "prev" §p34

The prev[p348] keyword may be used with link[p184], a[p267], area[p489], and form[p532] elements. This keyword creates a hyperlink[p313].

The prev[p348] 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[p348] keyword.

4.6.8.27 Other link types §p34

Extensions to the predefined set of link types may be registered on the microformats page for existing rel values. [MFREL][p1584]

Anyone is free to edit the microformats page for existing rel values 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[p184]
: One of the following:

  Not allowed
  : The keyword must not be specified on link[p184] elements.

  Hyperlink
  : The keyword may be specified on a link[p184] element; it creates a hyperlink[p313].

  External Resource
  : The keyword may be specified on a link[p184] element; it creates an external resource link[p313].

Effect on... a[p267] and area[p489]
: One of the following:

  Not allowed
  : The keyword must not be specified on a[p267] and area[p489] elements.

  Hyperlink
  : The keyword may be specified on a[p267] and area[p489] elements; it creates a hyperlink[p313].

  External Resource
  : The keyword may be specified on a[p267] and area[p489] elements; it creates an external resource link[p313].

  Hyperlink Annotation
  : The keyword may be specified on a[p267] and area[p489] elements; it annotates[p314] other hyperlinks[p313] created by the element.

Effect on... form[p532]

One of the following:

Not allowed
The keyword must not be specified on form[p532] elements.

Hyperlink
The keyword may be specified on form[p532] elements; it creates a hyperlink[p313].

External Resource
The keyword may be specified on form[p532] elements; it creates an external resource link[p313].

Hyperlink Annotation
The keyword may be specified on form[p532] elements; it annotates[p314] other hyperlinks[p313] 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 page for existing rel values 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 page for existing rel values with the status "proposed" or "ratified" may be used with the `rel` attribute on link[p184], a[p267], and area[p489] elements in accordance to the "Effect on..." field. [MFREL][p1584]

4.7 Edits §p35 0

The ins p350 and del p351 elements represent edits to the document.

4.7.1 The ins element §p35 0

Categories p154:

Flow content p157.
Phrasing content p157.
Palpable content p158.

Contexts in which this element can be used p154:

Where phrasing content p157 is expected.

Content model p154:

Transparent p159.

Tag omission in text/html p154:

Neither tag is omissible.

Content attributes p154:

Global attributes p162
cite p352 — Link to the source of the quotation or more information about the edit
datetime p352 — Date and (optionally) time of the change

Accessibility considerations p154:

For authors.
For implementers.

Sanitization p154:

Default p1249 with attributes cite p352, datetime p352.

DOM interface p154:

Uses HTMLModElement p352.

The ins p350 element represents p148 an addition to the document.

Example

The following represents the addition of a single paragraph:

```
<aside>
 <ins>
  <p> I like fruit. </p>
 </ins>
</aside>
```

As does the following, because everything in the aside p222 element here counts as phrasing content p157 and therefore there is just one paragraph p159:

```
<aside>
 <ins>
  Apples are <em>tasty</em>.
 </ins>
 <ins>
  So are pears.
 </ins>
</aside>
```

ins p350 elements should not cross implied paragraph p159 boundaries.

Example

The following example represents the addition of two paragraphs, the second of which was inserted in two parts. The first ins element in this example thus crosses a paragraph boundary, which is considered poor form.

```
<aside>
 <!-- don't do this -->
 <ins datetime="2005-03-16 00:00Z">
  <p> I like fruit. </p>
  Apples are <em>tasty</em>.
 </ins>
 <ins datetime="2007-12-19 00:00Z">
  So are pears.
 </ins>
</aside>
```

Here is a better way of marking this up. It uses more elements, but none of the elements cross implied paragraph boundaries.

```
<aside>
 <ins datetime="2005-03-16 00:00Z">
  <p> I like fruit. </p>
 </ins>
 <ins datetime="2005-03-16 00:00Z">
  Apples are <em>tasty</em>.
 </ins>
 <ins datetime="2007-12-19 00:00Z">
  So are pears.
 </ins>
</aside>
```

4.7.2 The del 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.

Sanitization:

Default with attributes cite, datetime.

DOM interface:

Uses HTMLModElement.

The del element represents a removal from the document.

del[p351] elements should not cross implied paragraph[p159] 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.

```
<h1>To Do</h1>
<ul>
 <li>Empty the dishwasher</li>
 <li><del datetime="2009-10-11T01:25-07:00">Watch Walter Lewin's lectures</del></li>
 <li><del datetime="2009-10-10T23:38-07:00">Download more tracks</del></li>
 <li>Buy a printer</li>
</ul>
```

4.7.3 Attributes common to ins[p350] and del[p351] elements §[p35 2]

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[p352] attribute is present, it must be a valid URL potentially surrounded by spaces[p100] that explains the change. To obtain the corresponding citation link, the value of the attribute must be parsed[p101] 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[p352] attribute's value must be a valid date string with optional time[p97].

User agents must parse the datetime[p352] attribute according to the parse a date or time string[p97] algorithm. If that doesn't return a date[p87] or a global date and time[p91], then the modification has no associated timestamp (the value is non-conforming; it is not a valid date string with optional time[p97]). Otherwise, the modification is marked as having been made at the given date[p87] or global date and time[p91]. If the given value is a global date and time[p91], 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[p350] and del[p351] elements must implement the HTMLModElement[p352] interface:

IDL

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

  [CEReactions, ReflectURL] attribute USVString cite;
  [CEReactions, Reflect] attribute DOMString dateTime;
};
```

4.7.4 Edits and paragraphs §[p35 2]

This section is non-normative.

Since the ins[p350] and del[p351] elements do not affect paragraphing[p159], it is possible, in some cases where paragraphs are implied[p159] (without explicit p[p238] elements), for an ins[p350] or del[p351] element to span both an entire paragraph or other non-phrasing content[p157] elements and part of another paragraph. For example:

```
<section>
 <ins>
  <p>
```

```
   This is a paragraph that was inserted.
  </p>
  This is another paragraph whose first sentence was inserted
  at the same time as the paragraph above.
 </ins>
 This is a second sentence, which was there all along.
</section>
```

By only wrapping some paragraphs in p[p238] 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[p350] or del[p351] element (though this is very confusing, and not considered good practice):

```
<section>
 This is the first paragraph. <ins>This sentence was
 inserted.
 <p>This second paragraph was inserted.</p>
 This sentence was inserted too.</ins> This is the
 third paragraph in this example.
 <!-- (don't do this) -->
</section>
```

However, due to the way implied paragraphs[p159] 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[p350] or del[p351] element. You instead have to use one (or two) p[p238] element(s) and two ins[p350] or del[p351] elements, as for example:

```
<section>
 <p>This is the first paragraph. <del>This sentence was
 deleted.</del></p>
 <p><del>This sentence was deleted too.</del> That
 sentence needed a separate &lt;del&gt; element.</p>
</section>
```

Partly because of the confusion described above, authors are strongly encouraged to always mark up all paragraphs with the p[p238] element, instead of having ins[p350] or del[p351] elements that cross implied paragraphs[p159] boundaries.

4.7.5 Edits and lists §[p353]

This section is non-normative.

The content models of the ol[p247] and ul[p249] elements do not allow ins[p350] and del[p351] 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[p350] or del[p351] element can be wrapped around the contents of the li[p251] element. To indicate that an item has been replaced by another, a single li[p251] element can have one or more del[p351] elements followed by one or more ins[p350] 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.

```
<h1>Stop-ship bugs</h1>
<ol>
 <li><ins datetime="2008-02-12T15:20Z">Bug 225:
 Rain detector doesn't work in snow</ins></li>
 <li><del datetime="2008-03-01T20:22Z"><ins datetime="2008-02-14T12:02Z">Bug 228:
 Water buffer overflows in April</ins></del></li>
 <li><ins datetime="2008-02-16T13:50Z">Bug 230:
```

```
 Water heater doesn't use renewable fuels</ins></li>
 <li><del datetime="2008-02-20T21:15Z"><ins datetime="2008-02-16T14:25Z">Bug 232:
 Carbon dioxide emissions detected after startup</ins></del></li>
</ol>
```

Example

In the following example, a list that started with just fruit was replaced by a list with just colors.

```
<h1>List of <del>fruits</del><ins>colors</ins></h1>
<ul>
 <li><del>Lime</del><ins>Green</ins></li>
 <li><del>Apple</del></li>
 <li>Orange</li>
 <li><del>Pear</del></li>
 <li><ins>Teal</ins></li>
 <li><del>Lemon</del><ins>Yellow</ins></li>
 <li>Olive</li>
 <li><ins>Purple</ins></li>
</ul>
```

4.7.6 Edits and tables §[p354]

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[p350] and del[p351] 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[p350] or del[p351] elements (respectively).

Example

Here, a table's row has been added:

```
<table>
 <thead>
  <tr> <th> Game name           <th> Game publisher   <th> Verdict
 <tbody>
  <tr> <td> Diablo 2           <td> Blizzard         <td> 8/10
  <tr> <td> Portal             <td> Valve            <td> 10/10
  <tr> <td> <ins>Portal 2</ins> <td> <ins>Valve</ins> <td> <ins>10/10</ins>
</table>
```

Here, a column has been removed (the time at which it was removed is given also, as is a link to the page explaining why):

```
<table>
 <thead>
  <tr> <th> Game name           <th> Game publisher   <th> <del cite="/edits/r192"
datetime="2011-05-02 14:23Z">Verdict</del>
 <tbody>
  <tr> <td> Diablo 2           <td> Blizzard         <td> <del cite="/edits/r192"
datetime="2011-05-02 14:23Z">8/10</del>
  <tr> <td> Portal             <td> Valve            <td> <del cite="/edits/r192"
datetime="2011-05-02 14:23Z">10/10</del>
  <tr> <td> Portal 2           <td> Valve            <td> <del cite="/edits/r192"
datetime="2011-05-02 14:23Z">10/10</del>
```

```
</table>
```

Generally speaking, there is no good way to indicate more complicated edits (e.g. that a cell was removed, moving all subsequent cells up or to the left).

4.8 Embedded content §[p355]

4.8.1 The `picture` element §[p355]

Categories[p154]:

Flow content[p157].
Phrasing content[p157].
Embedded content[p158].
Palpable content[p158].

Contexts in which this element can be used[p154]:

Where embedded content[p158] is expected.

Content model[p154]:

Zero or more `source`[p355] elements, followed by one `img`[p359] element, optionally intermixed with script-supporting elements[p159].

Tag omission in text/html[p154]:

Neither tag is omissible.

Content attributes[p154]:

Global attributes[p162]

Accessibility considerations[p154]:

For authors.
For implementers.

Sanitization[p154]:

Uncategorized[p1249].

DOM interface[p154]:

IDL
```
[Exposed=Window]
interface HTMLPictureElement : HTMLElement {
  [HTMLConstructor] constructor();
};
```

The `picture`[p355] element is a container which provides multiple sources to its contained `img`[p359] 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[p148] its children.

Note

The `picture`[p355] element is somewhat different from the similar-looking `video`[p420] and `audio`[p425] elements. While all of them contain `source`[p355] elements, the `source`[p355] element's `src`[p357] attribute has no meaning when the element is nested within a `picture`[p355] element, and the resource selection algorithm is different. Also, the `picture`[p355] element itself does not display anything; it merely provides a context for its contained `img`[p359] element that enables it to choose from multiple URLs.

4.8.2 The `source` element §[p355]

Categories[p154]:

None.

Contexts in which this element can be used[p154]**:**

As a child of a `picture`[p355] element, before the `img`[p359] element.
As a child of a media element[p430], before any flow content[p157] or `track`[p427] elements.

Content model[p154]**:**

Nothing[p155].

Tag omission in text/html[p154]**:**

No end tag[p1359].

Content attributes[p154]**:**

Global attributes[p162]
`type`[p356] — Type of embedded resource
`media`[p356] — Applicable media
`src`[p357] (in `audio`[p425] or `video`[p420]) — Address of the resource
`srcset`[p356] (in `picture`[p355]) — Images to use in different situations, e.g., high-resolution displays, small monitors, etc.
`sizes`[p357] (in `picture`[p355]) — Image sizes for different page layouts
`width`[p495] (in `picture`[p355]) — Horizontal dimension
`height`[p495] (in `picture`[p355]) — Vertical dimension

Accessibility considerations[p154]**:**

For authors.
For implementers.

Sanitization[p154]**:**

Uncategorized[p1249].

DOM interface[p154]**:**

IDL

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

  [CEReactions, ReflectURL] attribute USVString src;
  [CEReactions, Reflect] attribute DOMString type;
  [CEReactions, Reflect] attribute USVString srcset;
  [CEReactions, Reflect] attribute DOMString sizes;
  [CEReactions, Reflect] attribute DOMString media;
  [CEReactions, Reflect] attribute unsigned long width;
  [CEReactions, Reflect] attribute unsigned long height;
};
```

The `source`[p355] element allows authors to specify multiple alternative source sets[p378] for `img`[p359] elements or multiple alternative media resources[p432] for media elements[p430]. It does not represent[p148] anything on its own.

The **`type`** attribute may be present. If present, the value must be a valid MIME type string.

The **`media`** attribute may also be present. If present, the value must contain a valid media query list[p100]. The user agent will skip to the next `source`[p355] element if the value does not match the environment[p100].

Note

The `media`[p356] attribute is only evaluated once during the resource selection algorithm[p436] for media elements[p430]. In contrast, when using the `picture`[p355] element, the user agent will react to changes in the environment[p390].

The remainder of the requirements depend on whether the parent is a `picture`[p355] element or a media element[p430]:

↪ **The `source`[p355] element's parent is a `picture`[p355] element**

The **`srcset`** attribute must be present, and is a srcset attribute[p375].

The `srcset`[p356] attribute contributes the image sources[p378] to the source set[p378], if the `source`[p355] element is selected.

If the `srcset`[p356] attribute has any image candidate strings[p375] using a width descriptor[p375], the `sizes`[p357] attribute may also be present. If, additionally, the following sibling `img`[p359] element does not allow auto-sizes[p361], the `sizes`[p357] attribute must be

present. The **sizes** attribute is a sizes attribute[p376], which contributes the source size[p378] to the source set[p378], if the source[p355] element is selected.

Note

If the img[p359] *element allows auto-sizes*[p361]*, then the* sizes[p357] *attribute can be omitted on previous sibling* source[p355] *elements. In such cases, it is equivalent to specifying* auto[p376]*.*

The source[p355] element supports dimension attributes[p495]. The img[p359] element can use the width[p495] and height[p495] attributes of a source[p355] element, instead of those on the img[p359] element itself, to determine its rendered dimensions and aspect-ratio, as defined in the Rendering section[p1510].

The type[p356] attribute gives the type of the images in the source set[p378], to allow the user agent to skip to the next source[p355] element if it does not support the given type.

Note

If the type[p356] *attribute is* not *specified, the user agent will not select a different* source[p355] *element if it finds that it does not support the image format after fetching it.*

When a source[p355] element has a following sibling source[p355] element or img[p359] element with a srcset[p360] attribute specified, it must have at least one of the following:

- A media[p356] 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[p356] attribute specified.

The src[p357] attribute must not be present.

↪ **The source[p355] element's parent is a media element[p430]**

The **src** attribute gives the URL of the media resource[p432]. The value must be a valid non-empty URL potentially surrounded by spaces[p100]. This attribute must be present.

The type[p356] attribute gives the type of the media resource[p432], to help the user agent determine if it can play this media resource[p432] before fetching it. The `codecs` parameter, which certain MIME types define, might be necessary to specify exactly how the resource is encoded. [RFC6381][p1586]

Note

Dynamically modifying a source[p355] *element's* src[p357] *or* type[p356] *attribute when the element is already inserted in a* video[p420] *or* audio[p425] *element will have no effect. To change what is playing, just use the* src[p433] *attribute on the media element*[p430] *directly, possibly making use of the* canPlayType()[p434] *method to pick from amongst available resources. Generally, manipulating* source[p355] *elements manually after the document has been parsed is an unnecessarily complicated approach.*

Example

The following list shows some examples of how to use the `codecs=` MIME parameter in the type[p356] attribute.

H.264 Constrained baseline profile video (main and extended video compatible) level 3 and Low-Complexity AAC audio in MP4 container

```
<source src='video.mp4' type='video/mp4; codecs="avc1.42E01E, mp4a.40.2"'>
```

H.264 Extended profile video (baseline-compatible) level 3 and Low-Complexity AAC audio in MP4 container

```
<source src='video.mp4' type='video/mp4; codecs="avc1.58A01E, mp4a.40.2"'>
```

H.264 Main profile video level 3 and Low-Complexity AAC audio in MP4 container

```
<source src='video.mp4' type='video/mp4; codecs="avc1.4D401E, mp4a.40.2"'>
```

H.264 'High' profile video (incompatible with main, baseline, or extended profiles) level 3 and Low-Complexity

AAC audio in MP4 container

```
<source src='video.mp4' type='video/mp4; codecs="avc1.64001E, mp4a.40.2"'>
```

MPEG-4 Visual Simple Profile Level 0 video and Low-Complexity AAC audio in MP4 container

```
<source src='video.mp4' type='video/mp4; codecs="mp4v.20.8, mp4a.40.2"'>
```

MPEG-4 Advanced Simple Profile Level 0 video and Low-Complexity AAC audio in MP4 container

```
<source src='video.mp4' type='video/mp4; codecs="mp4v.20.240, mp4a.40.2"'>
```

MPEG-4 Visual Simple Profile Level 0 video and AMR audio in 3GPP container

```
<source src='video.3gp' type='video/3gpp; codecs="mp4v.20.8, samr"'>
```

Theora video and Vorbis audio in Ogg container

```
<source src='video.ogv' type='video/ogg; codecs="theora, vorbis"'>
```

Theora video and Speex audio in Ogg container

```
<source src='video.ogv' type='video/ogg; codecs="theora, speex"'>
```

Vorbis audio alone in Ogg container

```
<source src='audio.ogg' type='audio/ogg; codecs=vorbis'>
```

Speex audio alone in Ogg container

```
<source src='audio.spx' type='audio/ogg; codecs=speex'>
```

FLAC audio alone in Ogg container

```
<source src='audio.oga' type='audio/ogg; codecs=flac'>
```

Dirac video and Vorbis audio in Ogg container

```
<source src='video.ogv' type='video/ogg; codecs="dirac, vorbis"'>
```

The `srcset`[p356] and `sizes`[p357] attributes must not be present.

The `source`[p355] HTML element insertion steps[p46], given *insertedNode*, are:

1. Let *parent* be *insertedNode*'s parent.
2. If *parent* is a media element[p430] that has no `src`[p433] attribute and whose `networkState`[p435] has the value `NETWORK_EMPTY`[p435], then invoke that media element[p430]'s resource selection algorithm[p436].
3. If *parent* is a `picture`[p355] element, then for each *child* of *parent*'s children, if *child* is an `img`[p359] element, then count this as a relevant mutation[p379] for *child*.

The `source`[p355] HTML element moving steps[p46], given *movedNode*, *isSubtreeRoot*, and *oldAncestor* are:

1. If *movedNode*'s parent is a media element[p430] that has no `src`[p433] attribute and whose `networkState`[p435] has the value `NETWORK_EMPTY`[p435], then invoke that media element[p430]'s resource selection algorithm[p436].
2. If *isSubtreeRoot* is true and *oldAncestor* is a `picture`[p355] element, then for each *child* of *oldAncestor*'s children: if *child* is an `img`[p359] element, then count this as a relevant mutation[p379] for *child*.

The `source`[p355] HTML element removing steps[p46], given *removedNode*, *isSubtreeRoot*, and *oldAncestor* are:

1. If *isSubtreeRoot* is true and *oldAncestor* is a `picture`[p355] element, then for each *child* of *oldAncestor*'s children: if *child* is an `img`[p359] element, then count this as a relevant mutation[p379] for *child*.

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[p1575] event on the last source[p355] element and trigger fallback behavior:

```
<script>
 function fallback(video) {
   // replace <video> with its contents
   while (video.hasChildNodes()) {
     if (video.firstChild instanceof HTMLSourceElement)
       video.removeChild(video.firstChild);
     else
       video.parentNode.insertBefore(video.firstChild, video);
   }
   video.parentNode.removeChild(video);
 }
</script>
<video controls autoplay>
 <source src='video.mp4' type='video/mp4; codecs="avc1.42E01E, mp4a.40.2"'>
 <source src='video.ogv' type='video/ogg; codecs="theora, vorbis"'
         onerror="fallback(parentNode)">
 ...
</video>
```

4.8.3 The img element §[p359]

Categories[p154]:

Flow content[p157].
Phrasing content[p157].
Embedded content[p158].
Form-associated element[p532].
If the element has a usemap[p491] or controls[p363] attribute: Interactive content[p158].
Palpable content[p158].

Contexts in which this element can be used[p154]:

Where embedded content[p158] is expected.
As a child of a picture[p355] element, after all source[p355] elements.

Content model[p154]:

Nothing[p155].

Tag omission in text/html[p154]:

No end tag[p1359].

Content attributes[p154]:

Global attributes[p162]
alt[p360] — Replacement text for use when images are not available
src[p360] — Address of the resource
srcset[p360] — Images to use in different situations, e.g., high-resolution displays, small monitors, etc.
sizes[p361] — Image sizes for different page layouts
crossorigin[p361] — How the element handles crossorigin requests
usemap[p491] — Name of image map[p491] to use
ismap[p363] — Whether the image is a server-side image map
controls[p363] — Show user agent controls
width[p495] — Horizontal dimension
height[p495] — Vertical dimension
referrerpolicy[p361] — Referrer policy for fetches initiated by the element
decoding[p361] — Decoding hint to use when processing this image for presentation
loading[p361] — Used when determining loading deferral
fetchpriority[p361] — Sets the priority for fetches initiated by the element

Accessibility considerations[p154]**:**

If the element has a non-empty `alt`[p360] attribute: for authors; for implementers.
Otherwise: for authors; for implementers.

Sanitization[p154]**:**

Uncategorized[p1249].

DOM interface[p154]**:**

IDL

```
[Exposed=Window,
 LegacyFactoryFunction=Image(optional unsigned long width, optional unsigned long height)]
interface HTMLImageElement : HTMLElement {
  [HTMLConstructor] constructor();

  [CEReactions, Reflect] attribute DOMString alt;
  [CEReactions, ReflectURL] attribute USVString src;
  [CEReactions, Reflect] attribute USVString srcset;
  [CEReactions, Reflect] attribute DOMString sizes;
  [CEReactions] attribute DOMString? crossOrigin;
  [CEReactions, Reflect] attribute DOMString useMap;
  [CEReactions, Reflect] attribute boolean isMap;
  [CEReactions, Reflect] attribute boolean controls;
  [CEReactions, ReflectSetter] attribute unsigned long width;
  [CEReactions, ReflectSetter] 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;
  [CEReactions] attribute DOMString fetchPriority;

  Promise<undefined> decode();

  // also has obsolete members
};
```

An `img`[p359] element represents an image.

An `img`[p359] element has a **dimension attribute source**, initially set to the element itself.

The image given by the **`src`** and **`srcset`** attributes, and any previous sibling `source`[p355] elements' `srcset`[p356] attributes if the parent is a `picture`[p355] 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`[p359] element's fallback content[p158]).

The requirements on the `alt`[p360] attribute's value are described in a separate section[p391].

At least one of the `src`[p360] and `srcset`[p360] attributes must be present.

If the `src`[p360] attribute is present, it must contain a valid non-empty URL potentially surrounded by spaces[p100] referencing a non-interactive, optionally animated, image resource that is neither paged nor scripted.

Note

The requirements above imply that images can be static bitmaps (e.g. PNGs, GIFs, JPEGs), single-page vector documents (single-page PDFs, XML files with an SVG document element), animated bitmaps (APNGs, animated GIFs), animated vector graphics (XML files with an SVG document element that use declarative SMIL animation), and so forth. However, these definitions preclude SVG files with script, multipage PDF files, interactive MNG files, HTML documents, plain text documents, and the like. [PNG][p1585] [GIF][p1582] [JPEG][p1583] [PDF][p1585] [XML][p1588] [APNG][p1579] [SVG][p1586] [MNG][p1584]

The `srcset`[p360] attribute is a srcset attribute[p375].

The srcset attribute and the src attribute (if width descriptors are not used) contribute the image sources to the source set (if no source element was selected).

If the srcset attribute is present and has any image candidate strings using a width descriptor, the sizes attribute must also be present. If the srcset attribute is *not* specified, and the loading attribute is in the Lazy state, the sizes attribute may be specified with the value "auto" (ASCII case-insensitive). The **sizes** attribute is a sizes attribute, which contributes the source size to the source set (if no source element was selected).

An img element **allows auto-sizes** if:

- its loading attribute is in the Lazy state, and
- its sizes attribute's value is "auto" (ASCII case-insensitive), or starts with "auto," (ASCII case-insensitive).

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. [REFERRERPOLICY]

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 **fetchpriority** attribute is a fetch priority attribute. Its purpose is to set the priority used when fetching the image.

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

```
<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">
```

In the example above, the images load as follows:

↪ **1.jpeg**, **2.jpeg**, **4.jpeg**
The images load eagerly and delay the window's load event.

↪ **3.jpeg**
The image loads when layout is known, due to being in the viewport, however it does not delay the window's load event.

↪ **5.jpeg**
The image loads only once scrolled into the viewport, and does not delay the window's load event.

Note

Developers are encouraged to specify a preferred aspect ratio via width and height attributes on lazy loaded images, even if CSS sets the image's width and height properties, to prevent the page layout from shifting around after the image loads.

The img HTML element insertion steps, given *insertedNode*, are:

1. If *insertedNode*'s parent is a `picture`[p355] element, then, count this as a relevant mutation[p379] for *insertedNode*.

The `img`[p359] HTML element moving steps[p46], given *movedNode*, *isSubtreeRoot*, and *oldAncestor* are:

1. If *isSubtreeRoot* is true and *oldAncestor* is a `picture`[p355] element, then count this as a relevant mutation[p379] for *movedNode*.

The `img`[p359] HTML element removing steps[p46], given *removedNode*, *isSubtreeRoot*, and *oldAncestor* are:

1. If *isSubtreeRoot* is true and *oldAncestor* is a `picture`[p355] element, then count this as a relevant mutation[p379] for *removedNode*.
2. Set *removedNode*'s last auto-sizes width[p377] to null.

The `img`[p359] element must not be used as a layout tool. In particular, `img`[p359] 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`[p359] element represents depends on the `src`[p360] attribute and the `alt`[p360] attribute.

↪ **If the `src`[p360] attribute is set and the `alt`[p360] 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[p377] and the user agent is configured to display that image, then the element represents[p148] the element's image data.

Otherwise, the element represents[p148] 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`[p360] attribute is set and the `alt`[p360] attribute is set to a value that isn't empty**

The image is a key part of the content; the `alt`[p360] attribute gives a textual equivalent or replacement for the image.

If the image is available[p377] and the user agent is configured to display that image, then the element represents[p148] the element's image data.

Otherwise, the element represents[p148] the text given by the `alt`[p360] 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`[p360] attribute is set and the `alt`[p360] 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`[p360] 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[p377] and the user agent is configured to display that image, then the element represents[p148] the element's image data.

If the image has a `src`[p360] attribute whose value is the empty string, then the element represents[p148] 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`[p164] 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`[p259] element that has a child `figcaption`[p262] element, and, ignoring the `figcaption`[p262] element and its descendants, the `figure`[p259] element has no flow content[p157] descendants other than inter-element whitespace[p155] and the `img`[p359] element, then return the contents of the first such `figcaption`[p262] 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 **`controls`** attribute is a boolean attribute. If present, it indicates that the user agent may expose a user interface to the user. The attribute must not be specified on an element that does not have an `alt` attribute, or whose `alt` attribute's value is the empty string.

If the `controls` attribute is present, the user agent may expose controls over the image (e.g., a control for fullscreen viewing). The specific controls provided are implementation-defined, and can be platform-specific or based on the user's preferences.

If the user agent exposes a user interface by displaying controls over the `img` element, then the user agent should suppress any user interaction events while the user agent is interacting with this interface.

Issue #12318 tracks the interaction between image controls and animated images. User agents should not expose animation controls for images before that issue is resolved.

The `img` element supports dimension attributes.

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 **`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.

The **`fetchPriority`** IDL attribute must reflect the `fetchpriority` content attribute, limited to only known values.

For web developers (non-normative)

***image*.width [= *value*]**

***image*.height [= *value*]**

These attributes return the rendered dimensions of the image. When the image is not being rendered, they return the width and height content attributes' values, defaulting to naturalWidth and naturalHeight.

They can be set, to change the corresponding content attributes.

***image*.naturalWidth**

***image*.naturalHeight**

These attributes return the density-corrected natural width and height of the image, or 0 if the image is not *available*.

***image*.complete**

Returns true if the image has been completely downloaded or if no image is specified; otherwise, returns false.

***image*.currentSrc**

Returns the image's absolute URL.

***image*.decode()**

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.

***image* = new Image([*width* [, *height*]])**

Returns a new img element, with the width and height attributes set to the values passed in the relevant arguments, if applicable.

To determine the **dimensions** of an img element *image*:

1. Let *naturalDimensions* be null.
2. If *image* is *available*, then set *naturalDimensions* to *image*'s density-corrected natural width and height.
3. Return the result of determining the dimensions of *image* given *naturalDimensions*.

The **width** getter steps are to return the width of this's dimensions.

The **height** getter steps are to return the height of this's dimensions.

The **naturalWidth** and **naturalHeight** getter steps are:

1. If the image is not *available*, then return 0.
2. Return the respective component of the image's density-corrected natural width and height, in CSS pixels. [CSS]

Note

Since the density-corrected natural width and height of an image take into account any orientation specified in its metadata, naturalWidth and naturalHeight reflect the dimensions after applying any rotation needed to correctly orient the image, regardless of the value of the 'image-orientation' property.

The **complete** getter steps are:

1. If any of the following are 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; or
   - the img element's current request's state is broken and its pending request is null,

   then return true.
2. Return false.

The **currentSrc** IDL attribute must return the img[p359] element's current request[p377]'s current URL[p377].

The **decode()** method, when invoked, must perform the following steps:

1. Let *promise* be a new promise.
2. Queue a microtask[p1195] to perform the following steps:

   Note

   This is done because updating the image data[p380] takes place in a microtask as well. Thus, to make code such as

   ```
   img.src = "stars.jpg";
   img.decode();
   ```

   properly decode `stars.jpg`*, we need to delay any processing by one microtask.*

   1. Let *global* be this's relevant global object[p1151].
   2. If any of the following are true:

      - this's node document is not fully active[p1051]; or
      - this's current request[p377]'s state[p377] is broken[p377],

      then reject *promise* with an "EncodingError" DOMException.
   3. Otherwise, in parallel[p44], wait for one of the following cases to occur, and perform the corresponding actions:

      ↪ **This img[p359] element's node document stops being fully active[p1051]**
      ↪ **This img[p359] element's current request[p377] changes or is mutated**
      ↪ **This img[p359] element's current request[p377]'s state[p377] becomes broken[p377]**
      Queue a global task[p1195] on the DOM manipulation task source[p1203] with *global* to reject *promise* with an "EncodingError" DOMException.

      ↪ **This img[p359] element's current request[p377]'s state[p377] becomes completely available[p377]**
      Decode[p379] the image.

      If decoding does not need to be performed for this image (for example because it is a vector graphic) or the decoding process completes successfully, then queue a global task[p1195] on the DOM manipulation task source[p1203] with *global* to resolve *promise* with undefined.

      If decoding fails (for example due to invalid image data), then queue a global task[p1195] on the DOM manipulation task source[p1203] with *global* to reject *promise* with an "EncodingError" DOMException.

      User agents should ensure that the decoded media data stays readily available until at least the end of the next successful update the rendering[p1198] step in the event loop[p1193]. 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[p377] 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()[p365] method, the process of loading an img[p359] element and then displaying it might look like the following:

```
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()` [p365] method:

```
const img = new Image();
img.src = "nebula.jpg";
img.decode().then(() => {
    document.body.appendChild(img);
}).catch(() => {
    document.body.appendChild(new Text("Could not load the nebula :("));
});
```

This latter form avoids the dropped frames of the original, by allowing the user agent to decode the image in parallel [p44], and only inserting it into the DOM (and thus causing it to be painted) once the decoding process is complete.

Example

Because the `decode()` [p365] method attempts to ensure that the decoded image data is available for at least one frame, it can be combined with the `requestAnimationFrame()` [p1281] API. This means it can be used with coding styles or frameworks that ensure that all DOM modifications are batched together as animation frame callbacks [p1281]:

```
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 legacy factory function is provided for creating `HTMLImageElement` [p360] 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 [p1151]'s associated `Document` [p966].
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` [p495]" and *width*.
4. If *height* is given, then set an attribute value for *img* using "`height` [p495]" and *height*.
5. Return *img*.

Example

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[p360] 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:

```
<p>I lived in <img src="carouge.svg" alt=""> Carouge.</p>
```

Here it is used as an icon representing the town:

```
<p>Home town: <img src="carouge.svg" alt="Carouge"></p>
```

Here it is used as part of a text on the town:

```
<p>Carouge has a coat of arms.</p>
<p><img src="carouge.svg" alt="The coat of arms depicts a lion, sitting in front of a tree."></p>
<p>It is used as decoration all over the town.</p>
```

Here it is used as a way to support a similar text where the description is given as well as, instead of as an alternative to, the image:

```
<p>Carouge has a coat of arms.</p>
<p><img src="carouge.svg" alt=""></p>
<p>The coat of arms depicts a lion, sitting in front of a tree.
It is used as decoration all over the town.</p>
```

Here it is used as part of a story:

```
<p>She picked up the folder and a piece of paper fell out.</p>
<p><img src="carouge.svg" alt="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."></p>
<p>She stared at the folder. 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...</p>
```

Here it is not known at the time of publication what the image will be, only that it will be a coat of arms of some kind, and thus no replacement text can be provided, and instead only a brief caption for the image is provided, in the title[p164] attribute:

```
<p>The last user to have uploaded a coat of arms uploaded this one:</p>
<p><img src="last-uploaded-coat-of-arms.cgi" title="User-uploaded coat of arms."></p>
```

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 or 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.

```
<article>
 <h1>My cats</h1>
 <h2>Fluffy</h2>
 <p>Fluffy is my favorite.</p>
 <img src="fluffy.jpg" alt="She likes playing with a ball of yarn.">
 <p>She's just too cute.</p>
```

```
 <h2>Miles</h2>
 <p>My other cat, Miles just eats and sleeps.</p>
</article>
```

```
<article>
 <h1>Photography</h1>
 <h2>Shooting moving targets indoors</h2>
 <p>The trick here is to know how to anticipate; to know at what speed and
 what distance the subject will pass by.</p>
 <img src="fluffy.jpg" alt="A cat flying by, chasing a ball of yarn, can be
 photographed quite nicely using this technique.">
 <h2>Nature by night</h2>
 <p>To achieve this, you'll need either an extremely sensitive film, or
 immense flash lights.</p>
</article>
```

```
<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 §[p368]

4.8.4.1 Introduction §[p368]

This section is non-normative.

To embed an image in HTML, when there is only a single image resource, use the img[p359] element and its src[p360] attribute.

Example

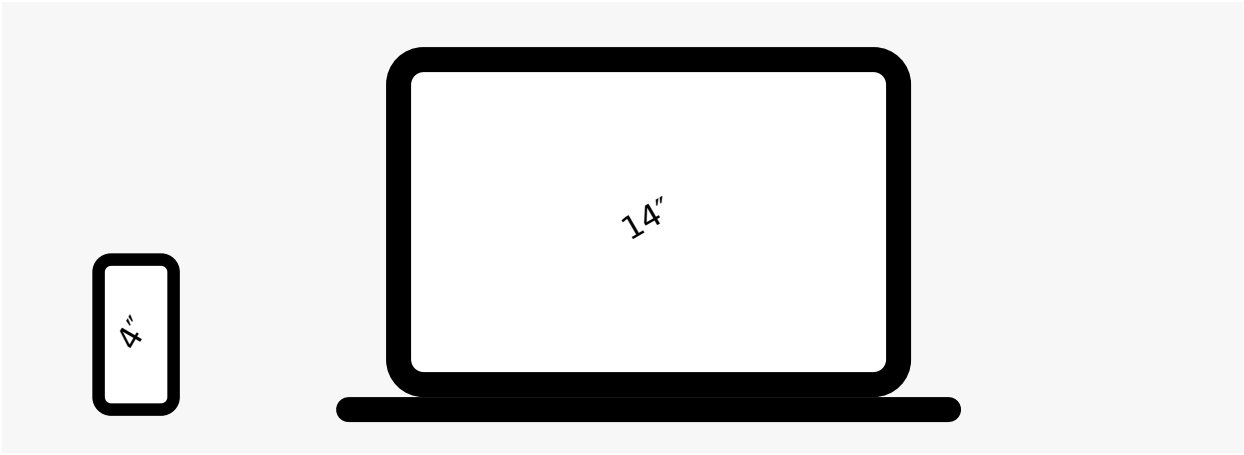
```
<h2>From today's featured article</h2>
<img src="/uploads/100-marie-lloyd.jpg" alt="" width="100" height="150">
<p><b><a href="/wiki/Marie_Lloyd">Marie Lloyd</a></b> (1870–1922)
was an English <a href="/wiki/Music_hall">music hall</a> singer, ...
```

However, there are a number of situations for which the author might wish to use multiple image resources that the user agent can choose from:

- Different users might have different environmental characteristics:
  - The users' physical screen size might be different from one another.

    Example

    A mobile phone's screen might be 4 inches diagonally, while a laptop's screen might be 14 inches diagonally.



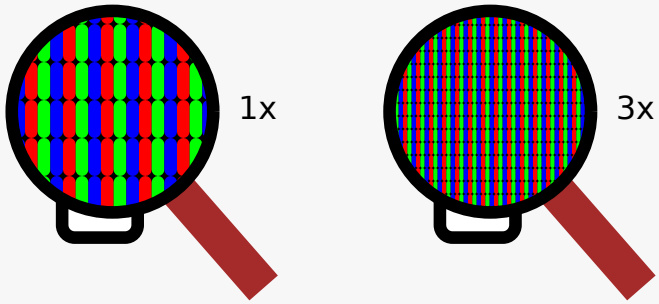
Note

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

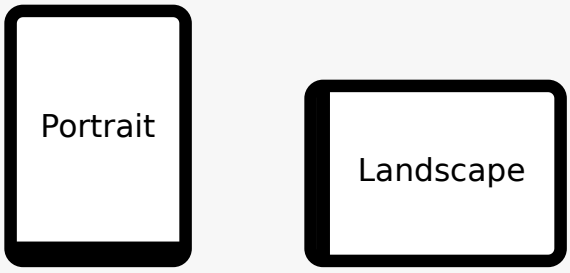
A user might zoom in to a particular image to be able to get a more detailed look.

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.

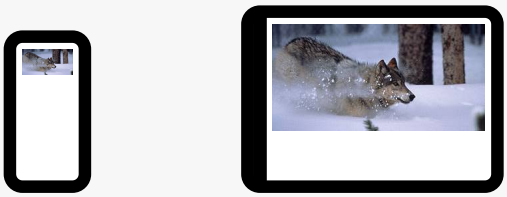
  Example

  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**.

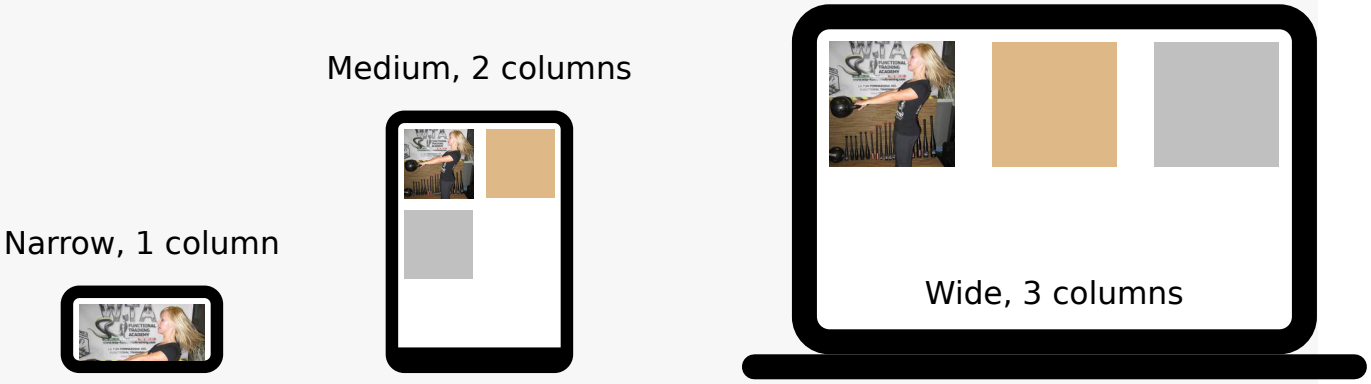
Example

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).



Example

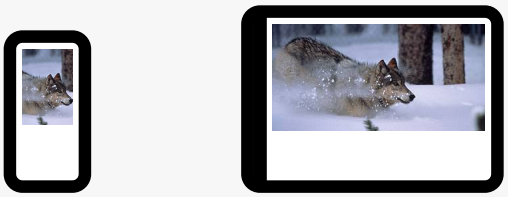
Another web page might have images in columns, with a single column for screens with a small physical size, two columns for screens with medium physical size, and three columns for screens with big physical size, with the images varying in rendered size in each case to fill up the viewport. In this case, 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 different image content depending on the rendered size of the image. This is usually referred to as **art direction**.

Example

When a web page is viewed on a screen with a large physical size (assuming a maximised browser window), the author might wish to include some less relevant parts surrounding the critical part of the image. When the same web page is viewed on a screen with a small physical size, the author might wish to show only the critical part of the image.



- Authors might want to show the same image content but using different image formats, depending on which image formats the user agent supports. This is usually referred to as **image format-based selection**.

  Example

  A web page might have some images in the JPEG, WebP and JPEG XR image formats, with the latter two having better compression abilities compared to JPEG. Since different user agents can support different image formats, with some formats offering better compression ratios, the author would like to serve the better formats to user agents that support them, while providing JPEG fallback for user agents that don't.

The above situations are not mutually exclusive. For example, it is reasonable to combine different resources for different device-pixel-ratio[p369] with different resources for art direction[p370].

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[p369]-based selection when the rendered size of the image is fixed

The src[p360] and srcset[p360] attributes on the img[p359] 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[p370]), but can be used together with art direction[p370].

Example

```
<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><b><a href="/wiki/Marie_Lloyd">Marie Lloyd</a></b> (1870–1922)
was an English <a href="/wiki/Music_hall">music hall</a> singer, ...
```

The user agent can choose any of the given resources depending on the user's screen's pixel density, zoom level, and possibly other factors such as the user's network conditions.

For backwards compatibility with older user agents that don't yet understand the srcset[p360] attribute, one of the URLs is specified in the img[p359] element's src[p360] 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[p360] attribute participates in the resource selection, as if it was specified in srcset[p360] with a 1x descriptor.

The image's rendered size is given in the width[p495] and height[p495] attributes, which allows the user agent to allocate space for

the image before it is downloaded.

Viewport-based selection[p370]

The srcset[p360] and sizes[p361] 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).

```
<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[p361] attribute. It can then choose any of the given resources depending on the user's screen's pixel density, zoom level, and possibly other factors such as the user's network conditions.

If the user's screen is 320 CSS pixels wide, this is equivalent to specifying `wolf-400.jpg 1.25x, wolf-800.jpg 2.5x, wolf-1600.jpg 5x`. On the other hand, if the user's screen is 1200 CSS pixels wide, this is equivalent to specifying `wolf-400.jpg 0.33x, wolf-800.jpg 0.67x, wolf-1600.jpg 1.33x`. By using the w descriptors and the sizes[p361] 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[p359] element's src[p360] attribute. In new user agents, the src[p360] attribute is ignored when the srcset[p360] 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.

```
<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[p361] 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[p361] 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.

Example

This example is the same as the previous example, but the image is lazy-loaded[p105]. In this case, the sizes[p361] attribute can use the auto[p376] keyword, and the user agent will use the width[p495] attribute (or the width specified in CSS) for the source size[p378].

```
<img loading="lazy" width="200" height="200" sizes="auto"
     srcset="swing-200.jpg 200w, swing-400.jpg 400w, swing-800.jpg 800w, swing-1600.jpg 1600w"
     src="swing-400.jpg" alt="Kettlebell Swing">
```

For better backwards-compatibility with legacy user agents that don't support the auto[p376] keyword, fallback sizes can be specified if desired.

```
<img loading="lazy" width="200" height="200"
     sizes="auto, (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">
```

Art direction[p370]-based selection

The picture[p355] element and the source[p355] element, together with the media[p356] attribute, can be used to provide multiple images that vary the image content (for instance the smaller image might be a cropped version of the bigger image).

Example

```
<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[p355] element for which the media query in the media[p356] attribute matches, and then choose an appropriate URL from its srcset[p356] 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[p370]- and device-pixel-ratio[p369]-based selection. A banner that takes half the viewport is provided in two versions, one for wide screens and one for narrow screens.

```
<h1>
 <picture>
  <source media="(max-width: 500px)" srcset="banner-phone.jpeg, banner-phone-HD.jpeg 2x">
  <img src="banner.jpeg" srcset="banner-HD.jpeg 2x" alt="The Breakfast Combo">
 </picture>
</h1>
```

Image format-based selection[p371]

The type[p356] attribute on the source[p355] element can be used to provide multiple images in different formats.

Example

```
<h2>From today's featured article</h2>
<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="" width="100" height="150">
</picture>
<p><b><a href="/wiki/Marie_Lloyd">Marie Lloyd</a></b> (1870–1922)
was an English <a href="/wiki/Music_hall">music hall</a> singer, ...
```

In this example, the user agent will choose the first source that has a type[p356] attribute with a supported MIME type. If the user agent supports WebP images, the first source[p355] element will be chosen. If not, but the user agent does support JPEG XR images, the second source[p355] element will be chosen. If neither of those formats are supported, the img[p359] element will be chosen.

4.8.4.1.1 Adaptive images §[p374]

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[p359] element's srcset[p360] attribute and the picture[p355] 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:

```
<figure>
 <picture>
  <source srcset="a-square.png" media="(max-width: 600px)">
  <img src="a-rectangle.png" alt="Barney Frank wears a suit and glasses.">
 </picture>
 <figcaption>Barney Frank, 2011</figcaption>
</figure>
```

Note

For details on what to put in the alt[p360] attribute, see the Requirements for providing text to act as an alternative for images[p391] 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:

```
<style>
 #a { width: 300px; height: 150px; }
 @media (max-width: 600px) { #a { width: 100px; height: 100px; } }
</style>
<figure>
 <picture>
  <source srcset="a-square.png" media="(max-width: 600px)">
  <img src="a-rectangle.png" alt="Barney Frank wears a suit and glasses." id="a">
 </picture>
 <figcaption>Barney Frank, 2011</figcaption>
</figure>
```

Alternatively, the width[p495] and height[p495] attributes can be used on the source[p355] and img[p359] elements to provide the width and height:

```
<figure>
 <picture>
  <source srcset="a-square.png" media="(max-width: 600px)" width="100" height="100">
  <img src="a-rectangle.png" width="300" height="150"
  alt="Barney Frank wears a suit and glasses.">
 </picture>
 <figcaption>Barney Frank, 2011</figcaption>
</figure>
```

The img[p359] element is used with the src[p360] attribute, which gives the URL of the image to use for legacy user agents that do not support the picture[p355] element. This leads to a question of which image to provide in the src[p360] attribute.

If the author wants the biggest image in legacy user agents, the markup could be as follows:

```
<picture>
 <source srcset="pear-mobile.jpeg" media="(max-width: 720px)">
 <source srcset="pear-tablet.jpeg" media="(max-width: 1280px)">
```

```
 <img src="pear-desktop.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[p355] elements, overriding the src[p360] attribute entirely.

```
<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[p360] attribute is actually being ignored entirely by picture[p355]-supporting user agents, the src[p360] attribute can default to any image, including one that is neither the smallest nor biggest:

```
<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.

```
<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[p355], img[p359], and link[p184] elements §[p37]

4.8.4.2.1 Srcset attributes §[p37]

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[p375], each separated from the next by a U+002C COMMA character (,). If an image candidate string[p375] contains no descriptors and no ASCII whitespace after the URL, the following image candidate string[p375], 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[p100] 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[p81] 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[p81] 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[p375] for an element that has the same width descriptor value[p375] as another image candidate string[p375]'s width descriptor value[p375] for the same element.

There must not be an image candidate string[p375] for an element that has the same pixel density descriptor value[p375] as another image candidate string[p375]'s pixel density descriptor value[p375] for the same element. For the purpose of this requirement, an image candidate string[p375] with no descriptors is equivalent to an image candidate string[p375] with a `1x` descriptor.

If an image candidate string[p375] for an element has the width descriptor[p375] specified, all other image candidate strings[p375] for that element must also have the width descriptor[p375] specified.

The specified width in an image candidate string[p375]'s width descriptor[p375] must match the natural width in the resource given by the image candidate string[p375]'s URL, if it has a natural width.

If an element has a sizes attribute[p376] present, all image candidate strings[p375] for that element must have the width descriptor[p375] specified.

4.8.4.2.2 Sizes attributes §[p376]

A **sizes attribute** is an attribute with requirements defined in this section.

If present, the value must be a valid source size list[p376].

A **valid source size list** is a string that matches the following grammar: [CSSVALUES][p1581] [MQ][p1584]

```
<source-size-list> = <source-size>#? , <source-size-value>
<source-size> = <media-condition> <source-size-value> | auto
<source-size-value> = <length> | auto
```

A <source-size-value>[p376] that is a <length> must not be negative, and must not use CSS functions other than the math functions.

The keyword **auto** is a width that is computed in parse a sizes attribute[p389]. If present, it must be the first entry and the entire <source-size-list>[p376] value must either be the string "`auto`" (ASCII case-insensitive) or start with the string "`auto,`" (ASCII case-insensitive).

Note

If the `img`[p359] *element that initiated the image loading (with the update the image data[p380] or react to environment changes[p390] algorithms) allows auto-sizes[p361] and is being rendered[p1488], then* `auto`[p376] *is the concrete object size width. If it allows auto-sizes[p361] and is not being rendered[p1488], but its last auto-sizes width[p377] is not null, then* `auto`[p376] *is that width. Otherwise, the* `auto`[p376] *value is ignored and the next source size[p378] is used instead, if any.*

The `auto`[p376] keyword may be specified in the `sizes`[p357] attribute of `source`[p355] elements and `sizes`[p361] attribute of `img`[p359] elements, if the following conditions are met. Otherwise, `auto`[p376] must not be specified.

- The element is a `source`[p355] element with a following sibling `img`[p359] element.
- The element is an `img`[p359] element.
- The `img`[p359] element referenced in either condition above allows auto-sizes[p361].

Note

In addition, it is strongly encouraged to specify dimensions using the `width`[p495] *and* `height`[p495] *attributes or with CSS. Without specified dimensions, the image will likely render with 300x150 dimensions because* `sizes="auto"` *implies* `contain-intrinsic-size: 300px 150px` *in the Rendering section[p1509].*

The <source-size-value>[p376] gives the intended layout width of the image. The author can specify different widths for different environments with <media-condition>s.

Note

Percentages are not allowed in a <source-size-value>[p376], to avoid confusion about what it would be relative to. The 'vw' unit can be used for sizes relative to the viewport width.

4.8.4.3 Processing model §[p377]

An `img`[p359] element has a **current request** and a **pending request**. The current request[p377] is initially set to a new image request[p377]. The pending request[p377] is initially set to null.

An **image request** has a **state**, **current URL**, and **image data**.

An image request[p377]'s state[p377] 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[p377]'s current URL[p377] is initially the empty string.

An image request[p377]'s image data[p377] is the decoded image data.

When an image request[p377]'s state[p377] is either partially available[p377] or completely available[p377], the image request[p377] is said to be **available**.

When an `img`[p359] element's current request[p377]'s state[p377] is completely available[p377] and the user agent can decode the media data without errors, then the `img`[p359] element is said to be **fully decodable**.

An image request[p377]'s state[p377] is initially unavailable[p377].

When an `img`[p359] element's current request[p377] is available[p377], the `img`[p359] element provides a paint source whose width is the image's density-corrected natural width[p377] (if any), whose height is the image's density-corrected natural height[p377] (if any), and whose appearance is the natural appearance of the image.

An `img`[p359] element is said to **use `srcset` or `picture`** if it has a `srcset`[p360] attribute specified or if it has a parent that is a `picture`[p355] element.

Each `img`[p359] element has a **last selected source**, which must initially be null.

Each `img`[p359] element has a **last auto-sizes width**, which is a length in CSS pixels or null, initially null.

Each image request[p377] has a **current pixel density**, which must initially be 1.

Each image request[p377] has **preferred density-corrected dimensions**, which is either a struct consisting of a width and a height or is null. It must initially be null.

To determine the **density-corrected natural width and height** of an `img`[p359] element *img*:

1. Let *density* be *img*'s current request[p377]'s current pixel density[p377].
2. Let *dimensions* be *img*'s current request[p377]'s preferred density-corrected dimensions[p377].

Note

The preferred density-corrected dimensions[p377] are set in the prepare an image for presentation[p384] algorithm based on meta information in the image.

3. If *dimensions* is not null, then set *dimensions*'s width to *dimensions*'s width divided by *density*, set *dimensions*'s height to *dimensions*'s height divided by *density*, and return *dimensions*.
4. Let *intrinsicWidth*, *intrinsicHeight*, and *intrinsicRatio* be *img*'s intrinsic width, intrinsic height, and intrinsic aspect ratio, if any, respectively.
5. If *intrinsicWidth* is not absent, then set *intrinsicWidth* to *intrinsicWidth* divided by *density*.
6. If *intrinsicHeight* is not absent, then set *intrinsicHeight* to *intrinsicHeight* divided by *density*.
7. Return the result of applying the default sizing algorithm with *intrinsicWidth*, *intrinsicHeight*, and *intrinsicRatio*, using a default object size of 300 by 150.

Example

For example, if the current pixel density[p377] is 3.125, that means that there are 300 device pixels per CSS inch, and thus if the image data is 300x600, it has density-corrected natural width and height[p377] of 96 CSS pixels by 192 CSS pixels.

All `img`[p359] and `link`[p184] elements are associated with a source set[p378].

A **source set** is an ordered set of zero or more image sources[p378] and a source size[p378].

An **image source** is a URL, and optionally either a pixel density descriptor[p375], or a width descriptor[p375].

A **source size** is a <source-size-value>[p376]. When a source size[p378] has a unit relative to the viewport, it must be interpreted relative to the `img`[p359] element's node document's viewport. Other units must be interpreted the same as in Media Queries. [MQ][p1584]

A **parse error** for algorithms in this section indicates a non-fatal mismatch between input and requirements. User agents are encouraged to expose parse error[p378]s 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[p102] 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[p102].

User agents must not support non-image resources with the `img`[p359] 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 §[p378]

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[p380] of the `img`[p359] element, with the *restart animation* flag set if so stated, whenever that element is created or has experienced relevant mutations[p379].

When obtaining images on demand, the user agent must update the image data[p380] of an `img`[p359] 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, 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 img or source HTML element insertion steps, HTML element removing steps, and HTML element moving steps count the mutation as a relevant mutation.
- The element's parent is a picture element and a source element that is a previous sibling has its srcset, sizes, media, type, width or height attributes set, changed, or removed.
- The element's adopting steps are run.
- If the element allows auto-sizes: the element starts being rendered, or its concrete object size width changes. This must set the *maybe omit events* flag for the update the image data algorithm.

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:

Keyword	State	Description
`sync`	**Sync**	Indicates a preference to decode this image synchronously for atomic presentation with other content.
`async`	**Async**	Indicates a preference to decode this image asynchronously to avoid delaying presentation of other content.
`auto`	**Auto**	Indicates no preference in decoding mode (the default).

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 *element*, optionally with the *restart animation* flag set, optionally with the *maybe omit events* flag set, it must run the following steps:

1. If *element*'s node document is not fully active:
    1. Continue running this algorithm in parallel.
    2. Wait until *element*'s node document is fully active.
    3. If another instance of this algorithm for *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 the current request's state to unavailable, set the pending request to null, and return.
3. Let *previousURL* be the current request's current URL.
4. Let *selected source* be null and *selected pixel density* be undefined.
5. If *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 *element*'s `src` attribute and set *selected pixel density* to 1.0.
6. Set *element*'s last selected source to *selected source*.

7. If *selected source* is not null:

    1. Let *urlString* be the result of encoding-parsing-and-serializing a URL[p101] given *selected source*, relative to *element*'s node document.

    2. If *urlString* is failure, then abort this inner set of steps.

    3. Let *key* be a tuple consisting of *urlString*, *element*'s `crossorigin`[p361] attribute's mode, and, if that mode is not No CORS[p103], *element*'s node document's origin.

    4. If the list of available images[p379] contains an entry for *key*:

        1. Set the ignore higher-layer caching[p379] flag for that entry.

        2. Abort the image request[p384] for the current request[p377] and the pending request[p377].

        3. Set the pending request[p377] to null.

        4. Set the current request[p377] to a new image request[p377] whose image data[p377] is that of the entry and whose state[p377] is completely available[p377].

        5. Prepare the current request for presentation[p384] given *element*.

        6. Set the current request[p377]'s current pixel density[p377] to *selected pixel density*.

        7. Queue an element task[p1195] on the DOM manipulation task source[p1203] given *element* and the following steps:

            1. If *restart animation* is set, then restart the animation[p1509].

            2. Set the current request[p377]'s current URL[p377] to *urlString*.

            3. If *maybe omit events* is not set or *previousURL* is not equal to *urlString*, then fire an event named `load`[p1575] at *element*.

        8. Abort the update the image data[p380] algorithm.

8. Queue a microtask[p1195] to perform the rest of this algorithm, allowing the task[p1194] that invoked this algorithm to continue.

9. If another instance of this algorithm for *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`[p360], `srcset`[p360], and `crossorigin`[p361] attributes are all set in succession.

10. Let *selected source* and *selected pixel density* be the URL and pixel density that results from selecting an image source[p385], respectively.

11. If *selected source* is null:

    1. Set the current request[p377]'s state[p377] to broken[p377], abort the image request[p384] for the current request[p377] and the pending request[p377], and set the pending request[p377] to null.

    2. Queue an element task[p1195] on the DOM manipulation task source[p1203] given *element* and the following steps:

        1. Change the current request[p377]'s current URL[p377] to the empty string.

        2. If all of the following are true:

            - *element* has a `src`[p360] attribute or it uses `srcset` or `picture`[p377]; and
            - *maybe omit events* is not set or *previousURL* is not the empty string,

            then fire an event named `error`[p1575] at *element*.

    3. Return.

12. Let *urlString* be the result of encoding-parsing-and-serializing a URL[p101] given *selected source*, relative to *element*'s node document.

13. If *urlString* is failure:

    1. Abort the image request[p384] for the current request[p377] and the pending request[p377].
    2. Set the current request[p377]'s state[p377] to broken[p377].
    3. Set the pending request[p377] to null.
    4. Queue an element task[p1195] on the DOM manipulation task source[p1203] given *element* and the following steps:

        1. Change the current request[p377]'s current URL[p377] to *selected source*.
        2. If *maybe omit events* is not set or *previousURL* is not equal to *selected source*, then fire an event named `error`[p1575] at *element*.

    5. Return.

14. If the pending request[p377] is not null and *urlString* is the same as the pending request[p377]'s current URL[p377], then return.

15. If *urlString* is the same as the current request[p377]'s current URL[p377] and the current request[p377]'s state[p377] is partially available[p377]:

    1. Abort the image request[p384] for the pending request[p377].
    2. If *restart animation* is set, then queue an element task[p1195] on the DOM manipulation task source[p1203] given *element* to restart the animation[p1509].
    3. Return.

16. Abort the image request[p384] for the pending request[p377].

17. Set *image request* to a new image request[p377] whose current URL[p377] is *urlString*.

18. If the current request[p377]'s state[p377] is unavailable[p377] or broken[p377], then set the current request[p377] to *image request*. Otherwise, set the pending request[p377] to *image request*.

19. Let *request* be the result of creating a potential-CORS request[p102] given *urlString*, "`image`", and the current state of *element*'s `crossorigin`[p361] content attribute.

20. Set *request*'s client to *element*'s node document's relevant settings object[p1151].

21. If *element* uses `srcset` or `picture`[p377], set *request*'s initiator to "`imageset`".

22. Set *request*'s referrer policy to the current state of *element*'s `referrerpolicy`[p361] attribute.

23. Set *request*'s priority to the current state of *element*'s `fetchpriority`[p361] attribute.

24. Let *delay load event* be true if the `img`[p359]'s lazy loading attribute[p105] is in the Eager[p105] state, or if scripting is disabled[p1152] for the `img`[p359], and false otherwise.

25. If the will lazy load element steps[p105] given the `img`[p359] return true:

    1. Set the `img`[p359]'s lazy load resumption steps[p105] to the rest of this algorithm starting with the step labeled *fetch the image*.
    2. Start intersection-observing a lazy loading element[p106] for *element*.
    3. Return.

26. *Fetch the image*: Fetch *request*. Return from this algorithm, and run the remaining steps as part of the fetch's *processResponse* for the response *response*.

    The resource obtained in this fashion, if any, is *image request*'s image data[p377]. It can be either CORS-same-origin[p102] or CORS-cross-origin[p102]; this affects the image's interaction with other APIs (e.g., when used on a `canvas`[p713]).

    When *delay load event* is true, fetching the image must delay the load event[p1458] of *element*'s node document until the task[p1194] that is queued[p1194] by the networking task source[p1204] once the resource has been fetched (defined below[p384]) 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[p939] 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.

27. As soon as possible, jump to the first applicable entry from the following list:

↪ **If the resource type is `multipart/x-mixed-replace`[p1547]**

The next task[p1194] that is queued[p1194] by the networking task source[p1204] while the image is being fetched must run the following steps:

1. If *image request* is the pending request[p377] and at least one body part has been completely decoded, abort the image request[p384] for the current request[p377], and upgrade the pending request to the current request[p384].
2. Otherwise, if *image request* is the pending request[p377] 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[p384] for the current request[p377], upgrade the pending request to the current request[p384], and set the current request[p377]'s state[p377] to broken[p377].
3. Otherwise, if *image request* is the current request[p377], its state[p377] is unavailable[p377], and the user agent is able to determine *image request*'s image's width and height, set the current request[p377]'s state[p377] to partially available[p377].
4. Otherwise, if *image request* is the current request[p377], its state[p377] is unavailable[p377], 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[p377]'s state[p377] to broken[p377].

Each task[p1194] that is queued[p1194] by the networking task source[p1204] while the image is being fetched must update the presentation of the image, but as each new body part comes in, if the user agent is able to determine the image's width and height, it must prepare *element*'s current request for presentation[p384] given *element* and replace the previous image. Once one body part has been completely decoded, perform the following steps:

1. Set *element*'s current request[p377]'s state[p377] to completely available[p377].
2. If *maybe omit events* is not set or *previousURL* is not equal to *urlString*, then queue an element task[p1195] on the DOM manipulation task source[p1203] given *element* to fire an event named `load`[p1575] at *element*.

↪ **If the resource type and data corresponds to a supported image format, as described below[p378]**

The next task[p1194] that is queued[p1194] by the networking task source[p1204] 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 the pending request[p377], set *image request*'s state[p377] to partially available[p377].
2. Otherwise, if the user agent is able to determine *image request*'s image's width and height, and *image request* is the current request[p377], prepare *image request* for presentation[p384] given *element* and set *image request*'s state[p377] to partially available[p377].
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 the pending request[p377]:
    1. Abort the image request[p384] for the current request[p377] and the pending request[p377].
    2. Upgrade the pending request to the current request[p384].
    3. Set the current request[p377]'s state[p377] to broken[p377].
    4. Fire an event named `error`[p1575] at *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 the current request[p377]:
    1. Abort the image request[p384] for *image request*.
    2. If *maybe omit events* is not set or *previousURL* is not equal to *urlString*, then fire an event named

error[p1575] at *element*.

That task[p1194], and each subsequent task[p1194] that is queued[p1194] by the networking task source[p1204] while the image is being fetched, must, if *image request* is the current request[p377], 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[p1194] that is queued[p1194] by the networking task source[p1204] once the resource has been fetched must additionally run these steps:

1. If *image request* is the pending request[p377], abort the image request[p384] for the current request[p377], upgrade the pending request to the current request[p384], and prepare *image request* for presentation[p384] given *element*.
2. Set *image request* to the completely available[p377] state.
3. Add the image to the list of available images[p379] using the key *key*, with the ignore higher-layer caching[p379] flag set.
4. If *maybe omit events* is not set or *previousURL* is not equal to *urlString*, then fire an event named load[p1575] at *element*.

↪ **Otherwise**

The image data is not in a supported file format; the user agent must set *image request*'s state[p377] to broken[p377], abort the image request[p384] for the current request[p377] and the pending request[p377], upgrade the pending request to the current request[p384] if *image request* is the pending request[p377], and then, if *maybe omit events* is not set or *previousURL* is not equal to *urlString*, queue an element task[p1195] on the DOM manipulation task source[p1203] given *element* to fire an event named error[p1575] at *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[p377] or null *image request* means to run the following steps:

1. If *image request* is null, then return.
2. Forget *image request*'s image data[p377], if any.
3. 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[p359] element means to run the following steps:

1. Set the img[p359] element's current request[p377] to the pending request[p377].
2. Set the img[p359] element's pending request[p377] to null.

4.8.4.3.6 Preparing an image for presentation §[p384]

To **prepare an image for presentation** for an image request[p377] *req* given image element *img*:

1. Let *exifTagMap* be the EXIF tags obtained from *req*'s image data[p377], as defined by the relevant codec. [EXIF][p1582]
2. Let *physicalWidth* and *physicalHeight* be the width and height obtained from *req*'s image data[p377], as defined by the relevant codec.
3. Let *dimX* be the value of *exifTagMap*'s tag `0xA002` (`PixelXDimension`).
4. Let *dimY* be the value of *exifTagMap*'s tag `0xA003` (`PixelYDimension`).
5. Let *resX* be the value of *exifTagMap*'s tag `0x011A` (`XResolution`).
6. Let *resY* be the value of *exifTagMap*'s tag `0x011B` (`YResolution`).
7. Let *resUnit* be the value of *exifTagMap*'s tag `0x0128` (`ResolutionUnit`).
8. If all the following are true:

- *dimX* is a positive integer;
- *dimY* is a positive integer;
- *resX* is a positive floating-point number;
- *resY* is a positive floating-point number;
- *physicalWidth* × 72 / *resX* is *dimX*;
- *physicalHeight* × 72 / *resY* is *dimY*;
- *resUnit* is `2` (`Inch`),

then:

1. If *req*'s image data[p377] is CORS-cross-origin[p102], then set *img*'s natural dimensions to *dimX* and *dimY*, and scale *img*'s pixel data accordingly.
2. Otherwise, set *req*'s preferred density-corrected dimensions[p377] to a struct with its width set to *dimX* and its height set to *dimY*.

9. Update *req*'s `img`[p359] element's presentation appropriately.

Note

Resolution in EXIF is equivalent to CSS points per inch, therefore 72 is the base for computing size from resolution.

It is not yet specified what would be the case if EXIF arrives after the image is already presented. See issue #4929.

4.8.4.3.7 Selecting an image source §p385

To **select an image source** given an `img`[p359] element *el*:

1. Update the source set[p386] for *el*.
2. If *el*'s source set[p378] is empty, return null as the URL and undefined as the pixel density.
3. Return the result of selecting an image[p385] from *el*'s source set[p378].

To **select an image source from a source set** given a source set[p378] *sourceSet*:

1. If an entry *b* in *sourceSet* has the same associated pixel density descriptor[p375] as an earlier entry *a* in *sourceSet*, then remove entry *b*. Repeat this step until none of the entries in *sourceSet* have the same associated pixel density descriptor[p375] as an earlier entry.
2. In an implementation-defined manner, choose one image source[p378] from *sourceSet*. Let *selectedSource* be this choice.
3. Return *selectedSource* and its associated pixel density.

4.8.4.3.8 Creating a source set from attributes §p385

When asked to **create a source set** given a string *default source*, a string *srcset*, a string *sizes*, and an element or null *img*:

1. Let *source set* be an empty source set[p378].
2. If *srcset* is not an empty string, then set *source set* to the result of parsing[p387] *srcset*.
3. Set *source set*'s source size[p378] to the result of parsing[p389] *sizes* with *img*.
4. If *default source* is not the empty string and *source set* does not contain an image source[p378] with a pixel density descriptor[p375] value of 1, and no image source[p378] with a width descriptor[p375], append *default source* to *source set*.
5. Normalize the source densities[p390] of *source set*.

6. Return *source set*.

4.8.4.3.9 Updating the source set §[p38 6]

When asked to **update the source set** for a given `img`[p359] or `link`[p184] element *el*, user agents must do the following:

1. Set *el*'s source set[p378] to an empty source set[p378].
2. Let *elements* be « *el* ».
3. If *el* is an `img`[p359] element whose parent node is a `picture`[p355] element, then replace the contents of *elements* with *el*'s parent node's child elements, retaining relative order.
4. Let *img* be *el* if *el* is an `img`[p359] element, otherwise null.
5. For each *child* in *elements*:
    1. If *child* is *el*:
        1. Let *default source* be the empty string.
        2. Let *srcset* be the empty string.
        3. Let *sizes* be the empty string.
        4. If *el* is an `img`[p359] element that has a `srcset`[p360] attribute, then set *srcset* to that attribute's value.
        5. Otherwise, if *el* is a `link`[p184] element that has an `imagesrcset`[p187] attribute, then set *srcset* to that attribute's value.
        6. If *el* is an `img`[p359] element that has a `sizes`[p361] attribute, then set *sizes* to that attribute's value.
        7. Otherwise, if *el* is a `link`[p184] element that has an `imagesizes`[p187] attribute, then set *sizes* to that attribute's value.
        8. If *el* is an `img`[p359] element that has a `src`[p360] attribute, then set *default source* to that attribute's value.
        9. Otherwise, if *el* is a `link`[p184] element that has an `href`[p185] attribute, then set *default source* to that attribute's value.
        10. Set *el*'s source set[p378] to the result of creating a source set[p385] given *default source*, *srcset*, *sizes*, and *img*.
        11. Return.

        > **Note**
        > *If* el *is a* `link`[p184] *element, then* elements *contains only* el*, so this step will be reached immediately and the rest of the algorithm will not run.*
    2. If *child* is not a `source`[p355] element, then continue.
    3. If *child* does not have a `srcset`[p356] attribute, continue to the next child.
    4. Parse *child*'s srcset attribute[p387] and let *source set* be the returned source set[p378].
    5. If *source set* has zero image sources[p378], continue to the next child.
    6. If *child* has a `media`[p356] attribute, and its value does not match the environment[p100], continue to the next child.
    7. Parse *child*'s sizes attribute[p389] with *img*, and let *source set*'s source size[p378] be the returned value.
    8. If *child* has a `type`[p356] attribute, and its value is an unknown or unsupported MIME type, continue to the next child.
    9. If *child* has `width`[p495] or `height`[p495] attributes, set *el*'s dimension attribute source[p360] to *child*. Otherwise, set *el*'s dimension attribute source[p360] to *el*.
    10. Normalize the source densities[p390] of *source set*.

11. Set *el*'s source set[p378] to *source set*.

12. Return.

Note

Each `img`[p359] *element independently considers its previous sibling* `source`[p355] *elements plus the* `img`[p359] *element itself for selecting an image source*[p378]*, ignoring any other (invalid) elements, including other* `img`[p359] *elements in the same* `picture`[p355] *element, or* `source`[p355] *elements that are following siblings of the relevant* `img`[p359] *element.*

4.8.4.3.10 Parsing a srcset attribute §[p387]

When asked to **parse a srcset attribute** from an element, parse the value of the element's srcset attribute[p375] 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[p378].
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[p378].
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 *url* be the result.
7. Let *descriptors* be a new empty list.
8. If *url* ends with U+002C (,):

    1. Remove all trailing U+002C COMMA characters from *url*. If this removed more than one character, that is a parse error[p378].

    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 PARENTHESIS (()**

        Append *c* to *current descriptor*. Set *state* to *in parens*.

        ↪ **EOF**

        If *current descriptor* is not empty, append *current descriptor* to *descriptors*. Jump to the step labeled *descriptor parser*.

        ↪ **Anything else**

        Append *c* to *current descriptor*.

↪ ***In parens***

Do the following, depending on the value of *c*:

↪ **U+0029 RIGHT PARENTHESIS ())**

Append *c* to *current descriptor*. Set *state* to *in descriptor*.

↪ **EOF**

Append *current descriptor* to *descriptors*. Jump to the step labeled *descriptor parser*.

↪ **Anything else**

Append *c* to *current descriptor*.

↪ ***After descriptor***

Do the following, depending on the value of *c*:

↪ **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[p81] followed by a U+0077 LATIN SMALL LETTER W character**

1. If the user agent does not support the `sizes`[p361] attribute, let *error* be *yes*.

Note

A conforming user agent will support the `sizes`[p361] *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[p81] to the descriptor. If the result is 0, let *error* be *yes*. Otherwise, let *width* be the result.

↪ **If the descriptor consists of a valid floating-point number[p81] 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[p82] to the descriptor. If the result is less than 0, let *error* be *yes*. Otherwise, let *density* be the result.

Note

If density *is 0, the natural dimensions will be infinite. User agents are expected to have limits in how big images can be rendered.*

↪ **If the descriptor consists of a valid non-negative integer[p81] followed by a U+0068 LATIN SMALL LETTER H character**

This is a parse error[p378].

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[p81] to the descriptor. If the result is 0, let *error* be *yes*. Otherwise, let *future-compat-h* be the result.

↪ **Anything else**

Let *error* be *yes*.

14. If *future-compat-h* is not *absent* and *width* is *absent*, let *error* be *yes*.
15. If *error* is still *no*, then append a new image source[p378] to *candidates* whose URL is *url*, associated with a width *width* if not *absent* and a pixel density *density* if not *absent*. Otherwise, there is a parse error[p378].
16. Return to the step labeled *splitting loop*.

4.8.4.3.11 Parsing a sizes attribute §[p389]

When asked to **parse a sizes attribute** from an element *element*, with an `img`[p359] element or null *img*:

1. Let *unparsed sizes list* be the result of parsing a comma-separated list of component values from the value of *element*'s sizes attribute[p376] (or the empty string, if the attribute is absent). [CSSSYNTAX][p1581]
2. Let *size* be null.
3. For each *unparsed size* in *unparsed sizes list*:
    1. Remove all consecutive <whitespace-token>s from the end of *unparsed size*. If *unparsed size* is now empty, then that is a parse error[p378]; continue.
    2. If the last component value in *unparsed size* is a valid non-negative <source-size-value>[p376], then set *size* to its value and remove the component value from *unparsed size*. Any CSS function other than the math functions is invalid. Otherwise, there is a parse error[p378]; continue.
    3. If *size* is `auto`[p376], and *img* is not null, and *img* allows auto-sizes[p361]:
        1. If *img* is being rendered[p1488], then set *size* to the concrete object size width of *img*, in CSS pixels, and set *img*'s last auto-sizes width[p377] to *size*.
        2. Otherwise, if *img*'s last auto-sizes width[p377] is not null, then set *size* to *img*'s last auto-sizes width[p377].

    Note

    If size *is still* `auto`[p376]*, then it will be ignored.*

    4. Remove all consecutive <whitespace-token>s from the end of *unparsed size*. If *unparsed size* is now empty:
        1. If this was not the last item in *unparsed sizes list*, that is a parse error[p378].
        2. If *size* is not `auto`[p376], then return *size*. Otherwise, continue.
    5. Parse the remaining component values in *unparsed size* as a <media-condition>. If it does not parse correctly, or it does parse correctly but the <media-condition> evaluates to false, continue. [MQ][p1584]
    6. If *size* is not `auto`[p376], then return *size*. Otherwise, continue.
4. Return `100vw`.

Note

It is invalid to use a bare <source-size-value>[p376] that is a <length> (without an accompanying <media-condition>) as an entry in the <source-size-list>[p376] that is not the last entry. However, the parsing algorithm allows it at any point in the <source-size-list>[p376], 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. A bare `auto`[p376] *keyword is allowed to have*

other entries following it to provide a fallback for legacy user agents.

4.8.4.3.12 Normalizing the source densities §p39 0

An image source[p378] can have a pixel density descriptor[p375], a width descriptor[p375], or no descriptor at all accompanying its URL. Normalizing a source set[p378] gives every image source[p378] a pixel density descriptor[p375].

When asked to **normalize the source densities** of a source set[p378] *source set*, the user agent must do the following:

1. Let *source size* be *source set*'s source size[p378].
2. For each image source[p378] in *source set*:
    1. If the image source[p378] has a pixel density descriptor[p375], continue to the next image source[p378].
    2. Otherwise, if the image source[p378] has a width descriptor[p375], replace the width descriptor[p375] with a pixel density descriptor[p375] with a value[p375] of the width descriptor value[p375] divided by *source size* and a unit of `x`.

        Note

        If the source size[p378] is 0, then the density would be infinity, which results in the natural dimensions being 0 by 0.

    3. Otherwise, give the image source[p378] a pixel density descriptor[p375] of `1x`.

4.8.4.3.13 Reacting to environment changes §p39 0

The user agent may at any time run the following algorithm to update an `img`[p359] 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`[p359] element is inserted into a document[p47], so that the density-corrected natural width and height[p377] match the new viewport, and so that the correct image is chosen when art direction[p370] is involved.

1. Await a stable state[p1201]. The synchronous section[p1201] consists of all the remaining steps of this algorithm until the algorithm says the synchronous section[p1201] has ended. (Steps in synchronous sections[p1201] are marked with ⌛.)
2. ⌛ If the `img`[p359] element does not use `srcset` or `picture`[p377], its node document is not fully active[p1051], it has image data whose resource type is `multipart/x-mixed-replace`[p1547], or its pending request[p377] 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[p385], 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[p377] and current pixel density[p377], then return.
6. ⌛ Let *urlString* be the result of encoding-parsing-and-serializing a URL[p101] given *selected source*, relative to the element's node document.
7. ⌛ If *urlString* is failure, then return.
8. ⌛ Let *corsAttributeState* be the state of the element's `crossorigin`[p361] content attribute.
9. ⌛ Let *origin* be the `img`[p359] element's node document's origin.
10. ⌛ Let *client* be the `img`[p359] element's node document's relevant settings object[p1151].

11. ⌛ Let *key* be a tuple consisting of *urlString*, *corsAttributeState*, and, if *corsAttributeState* is not No CORS, *origin*.

12. ⌛ Let *image request* be a new image request whose current URL is *urlString*.

13. ⌛ Set the element's pending request to *image request*.

14. End the synchronous section, continuing the remaining steps in parallel.

15. Let *finishUpdate* be the following steps:

    1. If the `img` element has experienced relevant mutations since this algorithm started, then set the pending request to null and abort these steps.

    2. Set the `img` element's last selected source to *selected source* and the `img` element's current pixel density to *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. Prepare *image request* for presentation given the `img` element.

    7. Fire an event named `load` at the `img` element.

16. If the list of available images contains an entry for *key*:

    1. Set *image request*'s image data to that of the entry.

    2. Queue an element task on the DOM manipulation task source given the `img` element and *finishUpdate*.

17. 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* and set *request*'s initiator to "`imageset`".

    3. Set *request*'s referrer policy to the current state of the element's `referrerpolicy` attribute.

    4. Set *request*'s priority to the current state of the element's `fetchpriority` attribute.

    5. Fetch *request*, with *processResponse* set to the following steps given response *response*:

        1. 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 set the pending request to null and return.

        2. 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 image's interaction with other APIs (e.g., when used on a `canvas`).

        3. Queue an element task on the DOM manipulation task source given the `img` element and *finishUpdate*.

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[p360] 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[p164] attribute can be used for supplemental information.

Another corollary is that the alt[p360] 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 §[p39]

When an a[p267] element that creates a hyperlink[p313], or a button[p585] element, has no textual content but contains one or more images, the alt[p360] 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:

```
<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>
```

Example

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.

```
<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:

```
<button name="rgb"><img src="red" alt="R"><img src="green" alt="G"><img src="blue"
alt="B"></button>
<button name="cmyk"><img src="cyan" alt="C"><img src="magenta" alt="M"><img src="yellow"
alt="Y"><img src="black" alt="K"></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:

```
<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 §p393

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 p359 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 p360 attribute, and must convey the same message as the image specified in the src p360 attribute.

It is important to realize that the alternative text is a *replacement* for the image, not a description of the image.

Example

In the following example we have a flowchart in image form, with text in the alt p360 attribute rephrasing the flowchart in prose form:

```
<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>
```

Example

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.

```
<!-- 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.

```
<!-- 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 p164 attribute or in the figcaption p262 element of a figure p259 with this image).

4.8.4.4.4 A short phrase or label with an alternative graphical representation: icons, logos §p393

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 p360 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[p360] attribute:

```
<nav>
 <p><a href="/help/"><img src="/icons/help.png" alt=""> Help</a></p>
 <p><a href="/configure/"><img src="/icons/configuration.png" alt="">
 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[p360] attribute.

Example

Here, posts on a news site are labeled with an icon indicating their topic.

```
<body>
 <article>
  <header>
   <h1>Ratatouille wins <i>Best Movie of the Year</i> award</h1>
   <p><img src="movies.png" alt="Movies"></p>
  </header>
  <p>Pixar has won yet another <i>Best Movie of the Year</i> award,
  making this its 8th win in the last 12 years.</p>
 </article>
 <article>
  <header>
   <h1>Latest TWiT episode is online</h1>
   <p><img src="podcasts.png" alt="Podcasts"></p>
  </header>
  <p>The latest TWiT episode has been posted, in which we hear
  several tech news stories as well as learning much more about the
  iPhone. This week, the panelists compare how reflective their
  iPhones' Apple logos are.</p>
 </article>
</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[p360] attribute must contain the name of the entity being represented by the logo. The alt[p360] 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[p360] 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:

```
<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:

```
<article>
 <h2>News</h2>
 <p>We have recently been looking at buying the <img src="alpha.gif"
 alt=""> AΒΓ company, a small Greek company
 specializing in our type of product.</p>
```

In this third snippet, we have a logo being used in an aside, as part of the larger article discussing the acquisition:

```
<aside><p><img src="alpha-large.gif" alt=""></p></aside>
 <p>The AΒΓ 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.

```
<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, oooooh, a question mark, how
<em>revolutionary</em>, how utterly <em>ground-breaking</em>, I'm
sure everyone will rush to adopt those specifications now! They could
at least have tried for some sort of, I don't know, sequence of
rounded squares with varying shades of green and bold white outlines,
at least that would look good on the cover of a blue book.</p>
```

This example shows how the alternative text should be written such that if the image isn't *available*[p377], and the text is used instead, the text flows seamlessly into the surrounding text, as if the image had never been there in the first place.

4.8.4.4.5 Text that has been rendered to a graphic for typographical effect §[p395]

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`[p360] 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:

```
<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.

```
<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.

```
<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 §[p39 6]

In many cases, the image is actually just supplementary, and its presence merely reinforces the surrounding text. In these cases, the `alt`[p360] 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`[p164] attribute can be used, or the `figure`[p259] and `figcaption`[p262] 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>
<p><img src="rendering-mode-pie-chart.png" alt=""></p>
```

4.8.4.4.7 Ancillary images §[p397]

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 `alt`[p360] 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 `alt`[p360] attribute's value must not be empty (as that would be quite confusing for non-visual readers).

Example

Consider a news article about a political figure, in which the individual's face was shown in an image. 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, decided by whether the image influences the interpretation of the prose.

In this first variant, the image is shown without context, and no alternative text is provided:

```
<p><img src="president.jpeg" alt=""> Ahead of today's referendum,
the President wrote an open letter to all registered voters. In it, she admitted that the country
was
divided.</p>
```

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:

```
<p><img src="president.jpeg" alt="The President is sad.">
Ahead of today's referendum, the President wrote an open letter to all
registered voters. In it, she admitted that the country was divided.
```

```
</p>
```

```
<p><img src="president.jpeg" alt="The President is happy!">
Ahead of today's referendum, the President wrote an open letter to all
registered voters. In it, she admitted that the country was divided.
</p>
```

Whether the individual was "sad" or "happy" makes a difference to how the rest of the paragraph is to be interpreted: is she likely saying that she is unhappy with the country being divided, or is she saying that the prospect of a divided country is good for her political career? The interpretation varies based on the image.

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.

```
<p>Ahead of today's referendum, the President wrote an open letter to
all registered voters. In it, she admitted that the country was divided.</p>
<figure>
 <img src="president.jpeg"
      alt="A high forehead, cheerful disposition, and dark hair round out the President's face.">
 <figcaption> The President of Ruritania. Photo © 2014 PolitiPhoto. </figcaption>
</figure>
```

4.8.4.4.8 A purely decorative image that doesn't add any information §p398

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 p359 element. Such images are decorative, but still form part of the content. In these cases, the alt p360 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>
```

4.8.4.4.9 A group of images that form a single larger picture with no links §p398

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 p360 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 p360 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 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 §[p39]

Generally, image maps[p491] 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[p360] attribute representing the purpose of the link.

Example

In the following example, a picture representing the flying spaghetti monster emblem, with each of the left noodly appendages and the right noodly appendages in different images, so that the user can pick the left side or the right side in an adventure.

```
<h1>The Church</h1>
<p>You come across a flying spaghetti monster. Which side of His
Noodliness do you wish to reach out for?</p>
<p><a href="?go=left" ><img src="fsm-left.png"  alt="Left side. "></a
  ><img src="fsm-middle.png" alt=""
  ><a href="?go=right"><img src="fsm-right.png" alt="Right side."></a></p>
```

4.8.4.4.11 A key part of the content §[p39]

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[p360] attribute.

Example

A screenshot in a gallery of screenshots for a new OS, with some alternative text:

```
<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:

```
<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[p360] attribute must contain some suitable alternative text, but it may be somewhat brief.

Example

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:

```
<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:

```
<!-- 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.

Example

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.

```
<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.">
```

Example

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:

```
<section class="bio">
 <h1>A Biography of Isaac Asimov</h1>
 <p>Born <b>Isaak Yudovich Ozimov</b> in 1920, Isaac was a prolific author.</p>
 <p><img src="headpics/asimov.jpeg" alt="Isaac Asimov had dark hair, a tall forehead, and wore
glasses.
 Later in life, he wore long white sideburns."></p>
 <p>Asimov was born in Russia, and moved to the US when he was three years old.</p>
 <p>...</p>
</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...".

Images whose contents are not known

In some unfortunate cases, there might be no alternative text available at all, either because the image is obtained in some automated fashion without any associated alternative text (e.g., a webcam), or because the page is being generated by a script 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`[p360] attribute may be omitted, but one of the following conditions must be met as well:

- The `img`[p359] element is in a `figure`[p259] element that contains a `figcaption`[p262] element that contains content other than inter-element whitespace[p155], and, ignoring the `figcaption`[p262] element and its descendants, the `figure`[p259] element has no flow content[p157] descendants other than inter-element whitespace[p155] and the `img`[p359] element.
- The `title`[p164] attribute is present and has a non-empty value.

Note

Relying on the `title`[p164] 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`[p360] 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:

```
<figure>
 <img src="1100670787_6a7c664aef.jpg">
 <figcaption>Bubbles traveled everywhere with us.</figcaption>
</figure>
```

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:

```
<article>
 <h1>I took a photo</h1>
 <p>I went out today and took a photo!</p>
 <figure>
  <img src="photo2.jpeg">
  <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:

```
<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>
```

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[p164] attribute):

```
<p><label>What does this image say?
<img src="captcha.cgi?id=8934" title="CAPTCHA">
<input type=text name=captcha></label>
(If you cannot see the image, you can use an <a
href="?audio">audio</a> test instead.)</p>
```

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:

```
<table>
 <thead>
  <tr> <th> Image <th> Description
 <tbody>
  <tr>
   <td> <img src="2421.png" title="Image 640 by 100, filename 'banner.gif'">
   <td> <input name="alt2421">
  <tr>
   <td> <img src="2422.png" title="Image 200 by 480, filename 'ad3.gif'">
   <td> <input name="alt2422">
</table>
```

Notice that even in this example, as much useful information as possible is still included in the title[p164] 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 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.

Markup generators should generally avoid using the image's own filename as the alternative text. Similarly, markup generators should avoid generating alternative text from any content that will be equally available to presentation user agents (e.g., web browsers).

Note

This is because once a page is generated, it will typically not be updated, whereas the browsers that later read the page can be updated by the user, therefore the browser is likely to have more up-to-date and finely-tuned heuristics than the markup generator did when generating the page.

4.8.4.4.15 Guidance for conformance checkers §p404

A conformance checker must report the lack of an `alt`p360 attribute as an error unless one of the conditions listed below applies:

- The `img`p359 element is in a `figure`p259 element that satisfies the conditions described abovep401.
- The `img`p359 element has a `title`p164 attribute with a value that is not the empty string (also as described abovep401).
- 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`p359 element has a (non-conforming) `generator-unable-to-provide-required-alt`p403 attribute whose value is the empty string. A conformance checker that is not reporting the lack of an `alt`p360 attribute as an error must also not report the presence of the empty `generator-unable-to-provide-required-alt`p403 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`p360 attribute as an error even in the presence of the `generator-unable-to-provide-required-alt`p403 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 §p404

✓ MDN

Categoriesp154**:**

Flow contentp157.
Phrasing contentp157.
Embedded contentp158.
Interactive contentp158.
Palpable contentp158.

Contexts in which this element can be usedp154**:**

Where embedded contentp158 is expected.

Content modelp154**:**

Nothingp155.

Tag omission in text/htmlp154**:**

Neither tag is omissible.

Content attributesp154**:**

Global attributesp162
`src`p405 — Address of the resource
`srcdoc`p405 — A document to render in the `iframe`p404
`name`p409 — Name of content navigablep1038
`sandbox`p409 — Security rules for nested content
`allow`p411 — Permissions policy to be applied to the `iframe`p404's contents
`allowfullscreen`p411 — Whether to allow the `iframe`p404's contents to use `requestFullscreen()`
`width`p495 — Horizontal dimension
`height`p495 — Vertical dimension
`referrerpolicy`p412 — Referrer policy for fetches initiated by the element
`loading`p412 — Used when determining loading deferral

Accessibility considerationsp154**:**

For authors.
For implementers.

Sanitization[p154]**:**

Unsafe[p1249].

DOM interface[p154]**:**

IDL

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

  [CEReactions, ReflectURL] attribute USVString src;
  [CEReactions] attribute (TrustedHTML or DOMString) srcdoc;
  [CEReactions, Reflect] attribute DOMString name;
  [SameObject, PutForwards=value, Reflect] readonly attribute DOMTokenList sandbox;
  [CEReactions, Reflect] attribute DOMString allow;
  [CEReactions, Reflect] attribute boolean allowFullscreen;
  [CEReactions, Reflect] attribute DOMString width;
  [CEReactions, Reflect] 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[p404] element represents[p148] its content navigable[p1038].

The **src** attribute gives the URL of a page that the element's content navigable[p1038] is to contain. The attribute, if present, must be a valid non-empty URL potentially surrounded by spaces[p100]. If the itemprop[p833] attribute is specified on an iframe[p404] element, then the src[p405] attribute must also be specified.

The **srcdoc** attribute gives the content of the page that the element's content navigable[p1038] is to contain. The value of the attribute is used to construct[p1081] **an iframe srcdoc document**, which is a Document[p135] whose URL matches about:srcdoc[p100].

The srcdoc[p405] attribute, if present, must have a value using the HTML syntax[p1356] that consists of the following syntactic components, in the given order:

1. Any number of comments[p1367] and ASCII whitespace.
2. Optionally, a DOCTYPE[p1356].
3. Any number of comments[p1367] and ASCII whitespace.
4. The document element, in the form of an html[p179] element[p1357].
5. Any number of comments[p1367] and ASCII whitespace.

Note

The above requirements apply in XML documents as well.

Example

Here a blog uses the srcdoc[p405] attribute in conjunction with the sandbox[p409] 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:

```
<article>
 <h1>I got my own magazine!</h1>
 <p>After much effort, I've finally found a publisher, and so now I
 have my own magazine! Isn't that awesome?! The first issue will come
 out in September, and we have articles about getting food, and about
 getting in boxes, it's going to be great!</p>
```

```
 <footer>
  <p>Written by <a href="/users/cap">cap</a>, 1 hour ago.
 </footer>
 <article>
  <footer> Thirteen minutes ago, <a href="/users/ch">ch</a> wrote: </footer>
  <iframe sandbox srcdoc="<p>did you get a cover picture yet?"></iframe>
 </article>
 <article>
  <footer> Nine minutes ago, <a href="/users/cap">cap</a> wrote: </footer>
  <iframe sandbox srcdoc="<p>Yeah, you can see it <a
href=&quot;/gallery?mode=cover&amp;amp;page=1&quot;>in my gallery</a>."></iframe>
 </article>
 <article>
  <footer> Five minutes ago, <a href="/users/ch">ch</a> wrote: </footer>
  <iframe sandbox srcdoc="<p>hey that's earl's table.
<p>you should get earl&amp;amp;me on the next cover."></iframe>
 </article>
```

Notice the way that quotes have to be escaped (otherwise the srcdoc[p405] 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[p405] attribute, and once more to prevent the ampersand from being misinterpreted when parsing the sandboxed content.

Furthermore, notice that since the DOCTYPE[p1356] is optional in iframe srcdoc documents[p405], and the html[p179], head[p180], and body[p212] elements have optional start and end tags[p1360], and the title[p181] element is also optional in iframe srcdoc documents[p405], the markup in a srcdoc[p405] attribute can be relatively succinct despite representing an entire document, since only the contents of the body[p212] element need appear literally in the syntax. The other elements are still present, but only by implication.

Note

In the HTML syntax[p1356], 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 sandbox[p409] attribute, to ensure safe embedding of content. (And remember to escape ampersands before quotation marks, to ensure quotation marks become " and not ".)

Note

In XML the U+003C LESS-THAN SIGN character (<) needs to be escaped as well. In order to prevent attribute-value normalization, some of XML's whitespace characters — specifically U+0009 CHARACTER TABULATION (tab), U+000A LINE FEED (LF), and U+000D CARRIAGE RETURN (CR) — also need to be escaped. [XML][p1588]

Note

If the src[p405] attribute and the srcdoc[p405] attribute are both specified together, the srcdoc[p405] attribute takes priority. This allows authors to provide a fallback URL for legacy user agents that do not support the srcdoc[p405] attribute.

The iframe[p404] HTML element post-connection steps[p46], given *insertedNode*, are:

1. If *insertedNode*'s node document's browsing context[p1046] is null, then return.
2. If *insertedNode* has a sandbox[p409] attribute, then parse the sandboxing directive[p958] given the attribute's value and *insertedNode*'s iframe sandboxing flag set[p959].
3. Create a new child navigable[p1039] for *insertedNode*.
4. Process the iframe attributes[p407] for *insertedNode*, with *initialInsertion*[p407] set to true.

The iframe[p404] HTML element removing steps[p46], given *removedNode*, are to destroy a child navigable[p1042] given *removedNode*.

Note

This happens without any unload[p1576] events firing (the element's content document[p1039] is destroyed[p1042], not unloaded[p1114]).

Although iframe[p404]s are processed while in a shadow tree, per the above, several other aspects of their behavior are not well-defined with regards to shadow trees. See issue #763 for more detail.

Whenever an iframe[p404] element with a non-null content navigable[p1038] has its srcdoc[p405] attribute set, changed, or removed, the user agent must process the iframe attributes[p407].

Similarly, whenever an iframe[p404] element with a non-null content navigable[p1038] but with no srcdoc[p405] attribute specified has its src[p405] attribute set, changed, or removed, the user agent must process the iframe attributes[p407].

To **process the iframe attributes** for an element *element*, with an optional boolean ***initialInsertion*** (default false):

1. If *element*'s srcdoc[p405] attribute is specified:
    1. Set *element*'s current navigation was lazy loaded[p409] boolean to false.
    2. If the will lazy load element steps[p105] given *element* return true:
        1. Set *element*'s lazy load resumption steps[p105] 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[p409] boolean to true.
        3. Start intersection-observing a lazy loading element[p106] for *element*.
        4. Return.
    3. *Navigate to the srcdoc resource*: Navigate an iframe or frame[p408] given *element*, about:srcdoc[p100], the empty string, and the value of *element*'s srcdoc[p405] attribute.

       The resulting Document[p135] must be considered an iframe srcdoc document[p405].
2. Otherwise:
    1. Let *url* be the result of running the shared attribute processing steps for iframe and frame elements[p407] given *element* and *initialInsertion*.
    2. If *url* is null, then return.
    3. If *url* matches about:blank[p100] and *initialInsertion* is true:
        1. Run the iframe load event steps[p408] given *element*.
        2. Return.
    4. Let *referrerPolicy* be the current state of *element*'s referrerpolicy[p412] content attribute.
    5. Set *element*'s current navigation was lazy loaded[p409] boolean to false.
    6. If the will lazy load element steps[p105] given *element* return true:
        1. Set *element*'s lazy load resumption steps[p105] to the rest of this algorithm starting with the step labeled *navigate*.
        2. Set *element*'s current navigation was lazy loaded[p409] boolean to true.
        3. Start intersection-observing a lazy loading element[p106] for *element*.
        4. Return.
    7. *Navigate*: Navigate an iframe or frame[p408] given *element*, *url*, *referrerPolicy*, null, and *initialInsertion*.

The **shared attribute processing steps for iframe and frame elements**, given an element *element* and a boolean *initialInsertion*, are:

1. Let *url* be the URL record about:blank[p54].

2. If *element* has a `src`[p405] attribute specified, and its value is not the empty string:

    1. Let *maybeURL* be the result of encoding-parsing a URL[p101] given that attribute's value, relative to *element*'s node document.

    2. If *maybeURL* is not failure, then set *url* to *maybeURL*.

3. If the inclusive ancestor navigables[p1041] of *element*'s node navigable[p1036] contains a navigable[p1035] whose active document[p1036]'s URL equals *url* with *exclude fragments* set to true, then return null.

4. If *url* matches `about:blank`[p100] and *initialInsertion* is true, then perform the URL and history update steps[p1078] given *element*'s content navigable[p1038]'s active document[p1036] and *url*.

    Note

    This is necessary in case url *is something like* `about:blank?foo`*. If* url *is just plain* `about:blank`*, this will do nothing.*

5. Return *url*.

To **navigate an `iframe` or `frame`** given an element *element*, a URL *url*, a referrer policy *referrerPolicy*, an optional string-or-null *srcdocString* (default null), and an optional boolean *initialInsertion* (default false):

1. Let *historyHandling* be "`auto`[p1062]".

2. If *element*'s content navigable[p1038]'s active document[p1036] is not completely loaded[p1114], then set *historyHandling* to "`replace`[p1062]".

3. If *element* is an `iframe`[p404]:

    1. Set *element*'s pending resource-timing start time[p409] to the current high resolution time given *element*'s node document's relevant global object[p1151].

    2. Set *element*'s pending resource-timing URL[p409] to *url*.

4. Navigate[p1063] *element*'s content navigable[p1038] to *url* using *element*'s node document, with *historyHandling*[p1063] set to *historyHandling*, *referrerPolicy*[p1063] set to *referrerPolicy*, *documentResource*[p1063] set to *srcdocString*, and *initialInsertion*[p1063] set to *initialInsertion*.

Each `Document`[p135] has an **iframe load in progress** flag and a **mute iframe load** flag. When a `Document`[p135] is created, these flags must be unset for that `Document`[p135].

To run the **iframe load event steps**, given an `iframe`[p404] element *element*:

1. Assert: *element*'s content navigable[p1038] is not null.

2. Let *childDocument* be *element*'s content navigable[p1038]'s active document[p1036].

3. If *childDocument* has its mute iframe load[p408] flag set, then return.

4. If *element*'s pending resource-timing start time[p409] is not null:

    1. Assert: *element*'s pending resource-timing URL[p409] is not null.

    2. Let *global* be *element*'s node document's relevant global object[p1151].

    3. Let *fallbackTimingInfo* be a new fetch timing info whose start time is *element*'s pending resource-timing start time[p409] and whose response end time is the current high resolution time given *global*.

    4. Mark resource timing given *fallbackTimingInfo*, the result of parsing[p101] *element*'s pending resource-timing URL[p409], "`iframe`[p404]", *global*, the empty string, a new response body info, and 0.

    5. Set *element*'s pending resource-timing start time[p409] to null.

    6. Set *element*'s pending resource-timing URL[p409] to null.

5. Set *childDocument*'s iframe load in progress[p408] flag.

6. Fire an event named `load`[p1575] at *element*.

7. Unset *childDocument*'s iframe load in progress[p408] flag.

⚠Warning!

This, in conjunction with scripting, can be used to probe the URL space of the local network's HTTP servers. User agents may implement cross-origin[p939] 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[p1458] of *element*'s node document if *element*'s content navigable[p1038] is non-null and any of the following are true:

- *element*'s content navigable[p1038]'s active document[p1036] is not ready for post-load tasks[p1459];
- *element*'s content navigable[p1038]'s is delaying load events[p1036] is true; or
- anything is delaying the load event[p1458] of *element*'s content navigable[p1038]'s active document[p1036].

Note

If, during the handling of the load[p1575] event, element's *content navigable[p1038] is again navigated[p1063], that will further delay the load event[p1458].*

Each iframe[p404] element has an associated **current navigation was lazy loaded** boolean, initially false. It is set and unset in the process the iframe attributes[p407] algorithm.

An iframe[p404] element whose current navigation was lazy loaded[p409] boolean is false potentially delays the load event[p409].

Each iframe[p404] element has an associated null or DOMHighResTimeStamp **pending resource-timing start time**, initially set to null.

Each iframe[p404] element has an associated null or URL **pending resource-timing URL**, initially set to null.

Note

If, when the element is created, the srcdoc[p405] attribute is not set, and the src[p405] attribute is either also not set or set but its value cannot be parsed[p101], the element's content navigable[p1038] will remain at the initial about:blank[p136] Document[p135].

Note

If the user navigates[p1063] away from this page, the iframe[p404]'s content navigable[p1038]'s active WindowProxy[p1036] object will proxy new Window[p965] objects for new Document[p135] objects, but the src[p405] attribute will not change.

The **name** attribute, if present, must be a valid navigable target name[p1043]. The given value is used to name the element's content navigable[p1038] if present when that is created[p1039].

The **sandbox** attribute, when specified, enables a set of extra restrictions on any content hosted by the iframe[p404]. Its value must be an unordered set of unique space-separated tokens[p99] that are ASCII case-insensitive. The allowed values are:

- allow-downloads[p959]
- allow-forms[p959]
- allow-modals[p959]
- allow-orientation-lock[p959]
- allow-pointer-lock[p959]
- allow-popups[p958]
- allow-popups-to-escape-sandbox[p959]
- allow-presentation[p959]
- allow-same-origin[p958]
- allow-scripts[p959]
- allow-top-navigation[p958]
- allow-top-navigation-by-user-activation[p958]
- allow-top-navigation-to-custom-protocols[p959]

When the attribute is set, the content is treated as being from a unique opaque origin[p939], forms, scripts, and various potentially annoying APIs are disabled, and links are prevented from targeting other navigables[p1035]. The allow-same-origin[p958] keyword causes the content to be treated as being from its real origin instead of forcing it into an opaque origin[p939]; the allow-top-navigation[p958] keyword allows the content to navigate[p1063] its traversable navigable[p1037]; the allow-top-navigation-by-user-activation[p958] keyword behaves similarly but allows such navigation[p1063] only when the browsing context's active window[p1036] has transient activation[p868]; the allow-top-navigation-to-custom-protocols[p959] reenables navigations toward non fetch scheme to be handed off

to external software[p1073]; and the allow-forms[p959], allow-modals[p959], allow-orientation-lock[p959], allow-pointer-lock[p959], allow-popups[p958], allow-presentation[p959], allow-scripts[p959], and allow-popups-to-escape-sandbox[p959] 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[p1046] respectively. The allow-downloads[p959] keyword allows content to perform downloads. [POINTERLOCK][p1585] [SCREENORIENTATION][p1586] [PRESENTATION][p1585]

The allow-top-navigation[p958] and allow-top-navigation-by-user-activation[p958] keywords must not both be specified, as doing so is redundant; only allow-top-navigation[p958] will have an effect in such non-conformant markup.

Similarly, the allow-top-navigation-to-custom-protocols[p959] keyword must not be specified if either allow-top-navigation[p958] or allow-popups[p958] are specified, as doing so is redundant.

⚠Warning!

Setting both the allow-scripts[p959] and allow-same-origin[p958] keywords together when the embedded page has the same origin[p940] as the page containing the iframe[p404] allows the embedded page to simply remove the sandbox[p409] attribute and then reload itself, effectively breaking out of the sandbox altogether.

⚠Warning!

These flags only take effect when the content navigable[p1038] of the iframe[p404] element is navigated[p1063]. Removing them, or removing the entire sandbox[p409] 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[p404] 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[p404]. 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[p409] attribute.

When an iframe[p404] element's sandbox[p409] attribute is set or changed while it has a non-null content navigable[p1038], the user agent must parse the sandboxing directive[p958] given the attribute's value and the iframe[p404] element's iframe sandboxing flag set[p959].

When an iframe[p404] element's sandbox[p409] attribute is removed while it has a non-null content navigable[p1038], the user agent must empty the iframe[p404] element's iframe sandboxing flag set[p959].

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).

```
<p>We're not scared of you! Here is your content, unedited:</p>
<iframe sandbox src="https://usercontent.example.net/getusercontent.cgi?id=12193"></iframe>
```

⚠Warning!

It is important to use a separate domain so that if the attacker convinces the user to visit that page directly, the page doesn't run in the context of the site's origin, which would make the user vulnerable to any attack found in the page.

Example

In this example, a gadget from another site is embedded. The gadget has scripting and forms enabled, and the origin sandbox restrictions are lifted, allowing the gadget to communicate with its originating server. The sandbox is still useful, however, as it disables plugins and popups, thus reducing the risk of the user being exposed to malware and other annoyances.

```
<iframe sandbox="allow-same-origin allow-forms allow-scripts"
        src="https://maps.example.com/embedded.html"></iframe>
```

Example

Suppose a file A contained the following fragment:

```
<iframe sandbox="allow-same-origin allow-forms" src=B></iframe>
```

Suppose that file B contained an iframe also:

```
<iframe sandbox="allow-scripts" src=C></iframe>
```

Further, suppose that file C contained a link:

```
<a href=D>Link</a>
```

For this example, suppose all the files were served as text/html[p1546].

Page C in this scenario has all the sandboxing flags set. Scripts are disabled, because the iframe[p404] in A has scripts disabled, and this overrides the allow-scripts[p959] keyword set on the iframe[p404] in B. Forms are also disabled, because the inner iframe[p404] (in B) does not have the allow-forms[p959] keyword set.

Suppose now that a script in A removes all the sandbox[p409] attributes in A and B. This would change nothing immediately. If the user clicked the link in C, loading page D into the iframe[p404] in B, page D would now act as if the iframe[p404] in B had the allow-same-origin[p958] and allow-forms[p959] keywords set, because that was the state of the content navigable[p1038] in the iframe[p404] in A when page B was loaded.

Generally speaking, dynamically removing or changing the sandbox[p409] 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[p136] for a Document[p135] in the iframe[p404]'s content navigable[p1038] is initialized. Its value must be a serialized permissions policy. [PERMISSIONSPOLICY][p1585]

Example

In this example, an iframe[p404] is used to embed a map from an online navigation service. The allow[p411] attribute is used to enable the Geolocation API within the nested context.

```
<iframe src="https://maps.example.com/" allow="geolocation"></iframe>
```

The **allowfullscreen** attribute is a boolean attribute[p79]. When specified, it indicates that Document[p135] objects in the iframe[p404] element's content navigable[p1038] will be initialized with a permissions policy[p136] which allows the "fullscreen" feature to be used from any origin[p939]. This is enforced by the process permissions policy attributes algorithm. [PERMISSIONSPOLICY][p1585]

Example

Here, an iframe[p404] is used to embed a player from a video site. The allowfullscreen[p411] attribute is needed to enable the player to show its video fullscreen.

```
<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>
 <iframe src="https://video.example.com/embed?id=92469812" allowfullscreen></iframe>
</article>
```

Note

Neither allow[p411] nor allowfullscreen[p411] can grant access to a feature in an iframe[p404] element's content navigable[p1038] 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* is not fully active, 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 content navigable's active document, the `allow` and `allowfullscreen` attributes only take effect when the content navigable 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 child navigable, 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 as well as allowing masking of some origins in the internal ancestor origin objects list creation steps. [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 **`srcdoc`** getter steps are:

✓ MDN

1. Let *attribute* be the result of getting an attribute by namespace and local name given null, `srcdoc`'s local name, and this.
2. If *attribute* is null, then return the empty string.
3. Return *attribute*'s value.

The `srcdoc` setter steps are:

1. Let *compliantString* be the result of invoking the get trusted type compliant string algorithm with `TrustedHTML`, this's relevant global object, the given value, "`HTMLIFrameElement srcdoc`", and "`script`".

2. Set an attribute value given this, srcdoc[p405]'s local name, and *compliantString*.

The supported tokens for sandbox[p405]'s DOMTokenList are the allowed values defined in the sandbox[p409] attribute and supported by the user agent.

The **referrerPolicy** IDL attribute must reflect[p108] the referrerpolicy[p412] content attribute, limited to only known values[p109].

The **loading** IDL attribute must reflect[p108] the loading[p412] content attribute, limited to only known values[p109].

The **contentDocument** getter steps are to return this's content document[p1039].

The **contentWindow** getter steps are to return this's content window[p1039].

Example

Here is an example of a page using an iframe[p404] to include advertising from an advertising broker:

```
<iframe src="https://ads.example.com/?customerid=923513721&amp;format=banner"
        width="468" height="60"></iframe>
```

4.8.6 The embed element §[p413]

Categories[p154]:

Flow content[p157].
Phrasing content[p157].
Embedded content[p158].
Interactive content[p158].
Palpable content[p158].

Contexts in which this element can be used[p154]:

Where embedded content[p158] is expected.

Content model[p154]:

Nothing[p155].

Tag omission in text/html[p154]:

No end tag[p1359].

Content attributes[p154]:

Global attributes[p162]
src[p414] — Address of the resource
type[p414] — Type of embedded resource
width[p495] — Horizontal dimension
height[p495] — Vertical dimension
Any other attribute that has no namespace (see prose).

Accessibility considerations[p154]:

For authors.
For implementers.

Sanitization[p154]:

Unsafe[p1249].

DOM interface[p154]:

IDL

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

  [CEReactions, ReflectURL] attribute USVString src;
  [CEReactions, Reflect] attribute DOMString type;
  [CEReactions, Reflect] attribute DOMString width;
```

```
  [CEReactions, Reflect] attribute DOMString height;
  Document? getSVGDocument();

  // also has obsolete members
};
```

The embed[p413] element provides an integration point for an external 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[p100].

If the itemprop[p833] attribute is specified on an embed[p413] element, then the src[p414] 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[p414] attribute and the src[p414] attribute are present, then the type[p414] attribute must specify the same type as the explicit Content-Type metadata[p102] of the resource given by the src[p414] attribute.

While any of the following conditions are occurring, any plugin[p48] instantiated for the element must be removed, and the embed[p413] element represents[p148] nothing:

- The element has neither a src[p414] attribute nor a type[p414] attribute.
- The element has a media element[p430] ancestor.
- The element has an ancestor object[p416] element that is *not* showing its fallback content[p158].

An embed[p413] 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[p1193] reached step 1[p1196].
- The element's node document is fully active[p1051].
- The element has either a src[p414] attribute set or a type[p414] attribute set (or both).
- The element's src[p414] attribute is either absent or its value is not the empty string.
- The element is not a descendant of a media element[p430].
- The element is not a descendant of an object[p416] element that is not showing its fallback content[p158].
- The element is being rendered[p1488], or was being rendered[p1488] the last time the event loop[p1193] reached step 1[p1196].

Whenever an embed[p413] element that was not potentially active[p414] becomes potentially active[p414], and whenever a potentially active[p414] embed[p413] element that is remaining potentially active[p414] has its src[p414] attribute set, changed, or removed or its type[p414] attribute set, changed, or removed, the user agent must queue an element task[p1195] on the **embed task source** given the element to run the embed element setup steps[p414] for that element.

The embed element setup steps for a given embed[p413] element *element* are as follows:

1. If another task[p1194] has since been queued to run the embed element setup steps[p414] for *element*, then return.
2. If *element* has a src[p414] attribute set:
    1. Let *url* be the result of encoding-parsing a URL[p101] given *element*'s src[p414] attribute's value, 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[p1151], destination is "`embed`", credentials mode is "`include`", mode is "`navigate`", initiator type is "`embed`", and whose use-URL-credentials flag is set.
    4. Fetch *request*, with *processResponse* set to the following steps given response *response*:
        1. If another task[p1194] has since been queued to run the embed element setup steps[p414] 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*.

   ↪ **Otherwise**

   1. If *element*'s content navigable is null, then create a new child navigable for *element*.
   2. Navigate *element*'s content navigable to *response*'s URL using *element*'s node document, with *response* set to *response*, and *historyHandling* set to "`replace`".

      Note

      element's `src` attribute does not get updated if the content navigable gets further navigated to other locations.

   3. *element* now represents its content navigable.

Fetching the resource must delay the load event of *element*'s node document.

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.

Note

It is intentional that the above algorithm allows response *to have a non-ok status. This allows servers to return data for plugins even with error responses (e.g., HTTP 500 Internal Server Error codes can still contain plugin data).*

To **display no plugin** for an `embed` element *element*:

1. Destroy a child navigable given *element*.
2. Display an indication that no plugin could be found for *element*, as the contents of *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.

The `embed` element potentially delays the load event.

The embed[p413] element supports dimension attributes[p495].

4.8.7 The object element §[p416]

Categories[p154]:

Flow content[p157].
Phrasing content[p157].
Embedded content[p158].
Listed[p532] form-associated element[p532].
Palpable content[p158].

Contexts in which this element can be used[p154]:

Where embedded content[p158] is expected.

Content model[p154]:

Transparent[p159].

Tag omission in text/html[p154]:

Neither tag is omissible.

Content attributes[p154]:

Global attributes[p162]
data[p417] — Address of the resource
type[p417] — Type of embedded resource
name[p417] — Name of content navigable[p1038]
form[p627] — Associates the element with a form[p532] element
width[p495] — Horizontal dimension
height[p495] — Vertical dimension

Accessibility considerations[p154]:

For authors.
For implementers.

Sanitization[p154]:

Unsafe[p1249].

DOM interface[p154]:

IDL

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

  [CEReactions, ReflectURL] attribute USVString data;
  [CEReactions, Reflect] attribute DOMString type;
  [CEReactions, Reflect] attribute DOMString name;
  readonly attribute HTMLFormElement? form;
  [CEReactions, Reflect] attribute DOMString width;
  [CEReactions, Reflect] 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 or as a child navigable.

The **data** attribute specifies the URL of the resource. It must be present, and must contain a valid non-empty URL potentially surrounded by spaces.

The **type** attribute, if present, specifies the type of the resource. If present, the attribute must be a valid MIME type string.

The **name** attribute, if present, must be a valid navigable target name. The given value is used to name the element's content navigable, if applicable, and if present when the element's content navigable 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,
- the element is not on the stack of open elements of an HTML parser or XML parser, and it is either inserted into a document or removed from a document,
- the element's node document changes whether it is fully active,
- one of the element's ancestor object elements changes to or from showing its fallback content,
- the element's classid attribute is set, changed, or removed,
- the element's classid attribute is not present, and its data attribute is set, changed, or removed,
- neither the element's classid attribute nor its data attribute are present, and its type attribute is set, changed, or removed,
- the element changes from being rendered to not being rendered, or vice versa,

...the user agent must queue 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 data attribute is present and its value is not the empty string:

   1. If the type attribute is present and its value is not a type that the user agent supports, then the user agent may jump to the step below labeled *fallback* without fetching the content to examine its real type.

   2. Let *url* be the result of encoding-parsing a URL given the data attribute's value, relative to the element's node document.

   3. If *url* is failure, then fire an event named error at the element and jump to the step below labeled *fallback*.

   4. Let *request* be a new request whose URL is *url*, client is the element's node document's relevant settings object, destination is "`object`", credentials mode is "`include`", mode is "`navigate`", initiator type is "`object`", and whose use-URL-credentials flag is set.

   5. Fetch *request*.

Fetching the resource must delay the load event[p1458] of the element's node document until the task[p1194] that is queued[p1194] by the networking task source[p1204] 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[p1194] that is queued[p1194] by the networking task source[p1204] 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`[p1575] 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[p102], then let the *resource type* be the type specified in the resource's Content-Type metadata[p102], 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 type, but the remote site overrides that and instead furnishes the user agent with a resource that triggers a different type of content with different security characteristics.

    3. Run the appropriate set of steps from the following list:

    ↪ If the resource has associated Content-Type metadata[p102]

        1. Let *binary* be false.

        2. If the type specified in the resource's Content-Type metadata[p102] 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[p102] 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[p102], and jump to the step below labeled *handler*.

        5. If there is a `type`[p417] attribute present on the `object`[p416] element, and its value is not `application/octet-stream`, then run the following steps:

            1. If 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`[p417] attribute.

            2. Jump to the step below labeled *handler*.

    ↪ Otherwise, if the resource does not have associated Content-Type metadata[p102]

        1. If there is a `type`[p417] attribute present on the `object`[p416] element, then let the *tentative type* be the type specified in that `type`[p417] 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*.

    4. 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[p48] supports, then let *resource type* be the type that that plugin can handle.

    Example

    For example, a plugin might say that it can handle resources with path components that end with the four character string "`.swf`".

Note

It is possible for this step to finish, or for one of the substeps above to jump straight to the next step, with resource type *still being unknown. In both cases, the next step will trigger fallback.*

9. *Handler*: Handle the content as given by the first of the following cases that matches:

   ↪ **If the *resource type* is an XML MIME type, or if the *resource type* does not start with "`image/`"**

   If the `object`[p416] element's content navigable[p1038] is null, then create a new child navigable[p1039] for the element.

   Let *response* be the response from fetch.

   If *response*'s URL does not match `about:blank`[p100], then navigate[p1063] the element's content navigable[p1038] to *response*'s URL using the element's node document, with *historyHandling*[p1063] set to "`replace`[p1062]".

   Note

   The `data`[p417] attribute of the `object`[p416] element doesn't get updated if the content navigable[p1038] gets further navigated[p1063] to other locations.

   The `object`[p416] element represents[p148] its content navigable[p1038].

   ↪ **If the *resource type* starts with "`image/`", and support for images has not been disabled**

   Destroy a child navigable[p1042] given the `object`[p416] element.

   Apply the image sniffing rules to determine the type of the image.

   The `object`[p416] element represents[p148] 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`[p416] element represents.

11. If the `object`[p416] element does not represent its content navigable[p1038], then once the resource is completely loaded, queue an element task[p1195] on the DOM manipulation task source[p1203] given the `object`[p416] element to fire an event named `load`[p1575] at the element.

    Note

    If the element does *represent its content navigable[p1038], then an analogous task will be queued when the created `Document`[p135] is completely finished loading[p1114].*

12. Return.

4. *Fallback*: The `object`[p416] element represents[p148] the element's children. This is the element's fallback content[p158]. Destroy a child navigable[p1042] given the element.

Due to the algorithm above, the contents of `object`[p416] elements act as fallback content[p158], used only when referenced resources can't be shown (e.g. because it returned a 404 error). This allows multiple `object`[p416] 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`[p416] element potentially delays the load event[p409].

The `form`[p627] attribute is used to explicitly associate the `object`[p416] element with its form owner[p626].

The `object`[p416] element supports dimension attributes[p495].

The **`contentDocument`** getter steps are to return this's content document[p1039].

The **contentWindow** getter steps are to return this's content window[p1039].

The willValidate[p654], validity[p654], and validationMessage[p656] attributes, and the checkValidity()[p656], reportValidity()[p656], and setCustomValidity()[p654] methods, are part of the constraint validation API[p653]. The form[p628] IDL attribute is part of the element's forms API.

Example

In this example, an HTML page is embedded in another using the object[p416] element.

```
<figure>
 <object data="clock.html"></object>
 <figcaption>My HTML Clock</figcaption>
</figure>
```

4.8.8 The video element §[p420]

Categories[p154]:

Flow content[p157].
Phrasing content[p157].
Embedded content[p158].
If the element has a controls[p481] attribute: Interactive content[p158].
Palpable content[p158].

Contexts in which this element can be used[p154]:

Where embedded content[p158] is expected.

Content model[p154]:

If the element has a src[p433] attribute: zero or more track[p427] elements, then transparent[p159], but with no media element[p430] descendants.
If the element does not have a src[p433] attribute: zero or more source[p355] elements, then zero or more track[p427] elements, then transparent[p159], but with no media element[p430] descendants.

Tag omission in text/html[p154]:

Neither tag is omissible.

Content attributes[p154]:

Global attributes[p162]
src[p433] — Address of the resource
crossorigin[p433] — How the element handles crossorigin requests
poster[p421] — Poster frame to show prior to video playback
preload[p446] — Hints how much buffering the media resource[p432] will likely need
autoplay[p452] — Hint that the media resource[p432] can be started automatically when the page is loaded
playsinline[p422] — Encourage the user agent to display video content within the element's playback area
loop[p450] — Whether to loop the media resource[p432]
muted[p483] — Whether to mute the media resource[p432] by default
controls[p481] — Show user agent controls
loading[p431] — Used when determining loading deferral
width[p495] — Horizontal dimension
height[p495] — Vertical dimension

Accessibility considerations[p154]:

For authors.
For implementers.

Sanitization[p154]:

Uncategorized[p1249].

DOM interface[p154]:

IDL
```
[Exposed=Window]
```

```
interface HTMLVideoElement : HTMLMediaElement {
  [HTMLConstructor] constructor();

  [CEReactions, Reflect] attribute unsigned long width;
  [CEReactions, Reflect] attribute unsigned long height;
  readonly attribute unsigned long videoWidth;
  readonly attribute unsigned long videoHeight;
  [CEReactions, ReflectURL] attribute USVString poster;
  [CEReactions, Reflect] 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 text can be shown to the users of these older browsers informing them of how to access the video contents.

Note

In particular, this content is not intended to address accessibility concerns. To make video content accessible to the partially sighted, the blind, the hard-of-hearing, the deaf, and those with other physical or cognitive disabilities, a variety of features are available. Captions can be provided, either embedded in the video stream or as external files using the track element. Sign-language tracks can be 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. [WEBVTT]

The video element is a media element whose media data is ostensibly video data, possibly with associated audio data.

The video element has an internal shadow tree with no slot elements.

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:
    1. Set the video element's poster lazy load resumption steps to null.
    2. There is no poster frame; return.
3. Let *url* be the result of encoding-parsing a URL given the poster attribute's value, relative to the element's node document.
4. If *url* is failure, then return. **Note** *There is no poster frame.*
5. Let *request* be a new request whose URL is *url*, client is the element's node document's relevant settings object, destination is "image", initiator type is "video", credentials mode is "include", and whose use-URL-credentials flag is set.
6. If the will lazy load element steps given the video return true:
    1. Let *posterResumptionSteps* be the rest of this algorithm starting with the step labeled *Fetch*.
    2. Set the video element's poster lazy load resumption steps to *posterResumptionSteps*.

3. Return.

7. Fetch *request*. This must delay the load event[p1458] of the element's node document.

8. If an image is thus obtained, the poster frame[p421] is that image. Otherwise, there is no poster frame[p421].

Note

The image given by the poster[p421] attribute, the poster frame[p421], is intended to be a representative frame of the video (typically one of the first non-blank frames) that gives the user an idea of what the video is like.

The **playsinline** attribute is a boolean attribute[p79]. 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.

Note

The absence of the playsinline[p422] attribute 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[p422] attribute has no effect.

If the poster[p421] attribute is present and the loading[p431] attribute is in the Lazy[p105] state, the user agent must defer loading the poster image source data until the element's lazy load resumption steps[p105] are invoked.

Example

```
<video src="1.mp4" poster="1.jpg" type="video/mp4">
<video src="2.mp4" type="video/mp4" loading="eager">
<video src="3.mp4" type="video/mp4" loading="lazy">
<video src="4.mp4" type="video/mp4" loading="lazy" autoplay>
<div id="very-large"></div> <!-- Everything after this div is below the viewport -->
<video src="5.mp4" type="video/mp4">
<video src="6.mp4" type="video/mp4" loading="lazy">
<video src="7.mp4" type="video/mp4" autoplay loading="lazy">
<video src="8.mp4" type="video/mp4" poster="8.jpg" loading="lazy">
<video src="9.mp4" type="video/mp4" preload="none" poster="9.jpg" loading="lazy">
<video src="10.mp4" type="video/mp4" preload="metadata" loading="lazy">
<video src="11.mp4" type="video/mp4" poster="11.jpg" preload="none" loading="eager">
```

In the example above, the videos load as follows:

↪ **1.mp4**
The video and poster image load eagerly and delay the window's load event.

↪ **2.mp4**, **5.mp4**
The videos load eagerly and delay the window's load event.

↪ **3.mp4**
The video loads when layout is known, due to being in the viewport, however it does not delay the window's load event.

↪ **4.mp4**
The video loads and autoplay playback begins when layout is known, due to being in the viewport, however it does not delay the window's load event.

↪ **6.mp4**
The video loads only once scrolled into the viewport, and does not delay the window's load event.

↪ **7.mp4**
The video loads and autoplay playback begins only once scrolled into the viewport, and does not delay the window's load event.

↪ **8.mp4**
The video and poster image load only once scrolled into the viewport, and does not delay the window's load event.

↪ `9.mp4`

The video does not load until played. The poster image loads only once scrolled into the viewport, and does not delay the window's load event.

↪ `10.mp4`

The video's metadata loads only once scrolled into the viewport, and does not delay the window's load event.

↪ `11.mp4`

The video does not load until played. The poster image loads eagerly and delays the window's load event.

A video[p420] element represents what is given for the first matching condition in the list below:

↪ **When no video data is available (the element's readyState[p452] attribute is either HAVE_NOTHING[p450], or HAVE_METADATA[p450] but no video data has yet been obtained at all, or the element's readyState[p452] attribute is any subsequent value but the media resource[p432] does not have a video channel)**

The video[p420] element represents[p148] its poster frame[p421], if any, or else transparent black with no natural dimensions.

↪ **When the video[p420] element is paused[p453], the current playback position[p448] is the first frame of video, and the element's show poster flag[p449] is set**

The video[p420] element represents[p148] its poster frame[p421], if any, or else the first frame of the video.

↪ **When the video[p420] element is paused[p453], and the frame of video corresponding to the current playback position[p448] is not available (e.g. because the video is seeking or buffering)**

↪ **When the video[p420] element is neither potentially playing[p453] nor paused[p453] (e.g. when seeking or stalled)**

The video[p420] element represents[p148] the last frame of the video to have been rendered.

↪ **When the video[p420] element is paused[p453]**

The video[p420] element represents[p148] the frame of video corresponding to the current playback position[p448].

↪ **Otherwise (the video[p420] element has a video channel and is potentially playing[p453])**

The video[p420] element represents[p148] the frame of video at the continuously increasing "current" position[p448]. When the current playback position[p448] changes such that the last frame rendered is no longer the frame corresponding to the current playback position[p448] in the video, the new frame must be rendered.

Frames of video must be obtained from the video track that was selected[p465] when the event loop[p1193] last reached step 1[p1196].

Note

Which frame in a video stream corresponds to a particular playback position is defined by the video stream's format.

The video[p420] element also represents[p148] any text track cues[p468] whose text track cue active flag[p469] is set and whose text track[p466] is in the showing[p467] mode, and any audio from the media resource[p432], at the current playback position[p448].

Any audio associated with the media resource[p432] must, if played, be played synchronized with the current playback position[p448], at the element's effective media volume[p482]. The user agent must play the audio from audio tracks that were enabled[p465] when the event loop[p1193] 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[p148] a link to an external video playback utility or to the video data itself.

When a video[p420] element's media resource[p432] has a video channel, the element provides a paint source whose width is the media resource[p432]'s natural width[p424], whose height is the media resource[p432]'s natural height[p424], and whose appearance is the frame of video corresponding to the current playback position[p448], 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.

For web developers (non-normative)

***video*.videoWidth**[p424]

***video*.videoHeight**[p424]

These attributes return the natural dimensions of the video, or 0 if the dimensions are not known.

The **natural width** and **natural height** of the media resource[p432] 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** getter steps are:

1. If this's readyState[p452] attribute is HAVE_NOTHING[p450], then return 0.
2. Return the natural width[p424] of the video in CSS pixels.

The **videoHeight** getter steps are:

1. If this's readyState[p452] attribute is HAVE_NOTHING[p450], then return 0.
2. Return the natural height[p424] of the video in CSS pixels.

Whenever the natural width[p424] or natural height[p424] of the video changes (including, for example, because the selected video track[p465] was changed), if the element's readyState[p452] attribute is not HAVE_NOTHING[p450], the user agent must queue a media element task[p432] given the media element[p430] to fire an event named resize[p485] at the media element[p430].

The video[p420] element supports dimension attributes[p495].

In the absence of style rules to the contrary, video content should be rendered inside the element's playback area such that the video content is shown centered in the playback area at the largest possible size that fits completely within it, with the video content's aspect ratio being preserved. Thus, if the aspect ratio of the playback area does not match the aspect ratio of the video, the video will be shown letterboxed or pillarboxed. Areas of the element's playback area that do not contain the video represent nothing.

Note

In user agents that implement CSS, the above requirement can be implemented by using the style rule suggested in the Rendering section[p1508].

The natural width of a video[p420] element's playback area is the natural width of the poster frame[p421], if that is available and the element currently represents[p148] its poster frame; otherwise, it is the natural width[p424] of the video resource, if that is available; otherwise the natural width is missing.

The natural height of a video[p420] element's playback area is the natural height of the poster frame[p421], if that is available and the element currently represents[p148] its poster frame; otherwise it is the natural height[p424] of the video resource, if that is available; otherwise the natural height is missing.

The default object size is a width of 300 CSS pixels and a height of 150 CSS pixels. [CSSIMAGES][p1581]

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[p422] 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[p481]. In such an independent viewing mode, however, user agents may make full user interfaces visible, even if the controls[p481] 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.

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>
```

4.8.9 The audio element §[p425]

Categories[p154]:

Flow content[p157].
Phrasing content[p157].
Embedded content[p158].
If the element has a controls[p481] attribute: Interactive content[p158].
If the element has a controls[p481] attribute: Palpable content[p158].

Contexts in which this element can be used[p154]:

Where embedded content[p158] is expected.

Content model[p154]:

If the element has a src[p433] attribute: zero or more track[p427] elements, then transparent[p159], but with no media element[p430] descendants.
If the element does not have a src[p433] attribute: zero or more source[p355] elements, then zero or more track[p427] elements, then transparent[p159], but with no media element[p430] descendants.

Tag omission in text/html[p154]:

Neither tag is omissible.

Content attributes[p154]:

Global attributes[p162]
src[p433] — Address of the resource
crossorigin[p433] — How the element handles crossorigin requests
preload[p446] — Hints how much buffering the media resource[p432] will likely need
autoplay[p452] — Hint that the media resource[p432] can be started automatically when the page is loaded
loop[p450] — Whether to loop the media resource[p432]
muted[p483] — Whether to mute the media resource[p432] by default
controls[p481] — Show user agent controls

loading[p431] — Used when determining loading deferral

Accessibility considerations[p154]:

For authors.
For implementers.

Sanitization[p154]:

Uncategorized[p1249].

DOM interface[p154]:

IDL

```
[Exposed=Window,
 LegacyFactoryFunction=Audio(optional DOMString src)]
interface HTMLAudioElement : HTMLMediaElement {
  [HTMLConstructor] constructor();
};
```

An audio[p425] element represents[p148] a sound or audio stream.

Content may be provided inside the audio[p425] element. User agents should not show this content to the user; it is intended for older web browsers which do not support audio[p425], so that text can be shown 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[p420] element can be used instead of the audio[p425] element to play the audio, allowing users to enable the visual alternatives. Chapter titles can be provided to aid navigation, using the track[p427] 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[p425] element. [WEBVTT][p1588]

The audio[p425] element is a media element[p430] whose media data[p431] is ostensibly audio data.

The audio[p425] element has an internal shadow tree with no slot[p712] elements.

The src[p433], crossorigin[p433], preload[p446], autoplay[p452], loop[p450], muted[p483], controls[p481], and loading[p431] attributes are the attributes common to all media elements[p431].

Note

audio[p425] elements without a controls[p481] attribute will not be displayed by the user agent, preventing them from lazy loading.

Example

```
<audio src="1.mp3" type="audio/mpeg" controls>
<audio src="2.mp3" type="audio/mpeg" controls loading="eager">
<audio src="3.mp3" type="audio/mpeg" controls loading="lazy">
<audio src="4.mp3" type="audio/mpeg" controls loading="lazy" autoplay>
<div id="very-large"></div> <!-- Everything after this div is below the viewport -->
<audio src="5.mp3" type="audio/mpeg" controls>
<audio src="6.mp3" type="audio/mpeg" controls loading="lazy">
<audio src="7.mp3" type="audio/mpeg" controls autoplay loading="lazy">
<audio src="8.mp3" type="audio/mpeg" controls preload="metadata" loading="lazy">
```

In the example above, the audio files load as follows:

↪ `1.mp3`, `2.mp3`, `5.mp3`
The audio file loads eagerly and delays the window's load event.

↪ `3.mp3`
The audio loads when layout is known, due to being in the viewport, however it does not delay the window's load event.

↪ 4.mp3
The audio loads and autoplay playback begins when layout is known, due to being in the viewport, however it does not delay the window's load event.

↪ 6.mp3
The audio loads only once scrolled into the viewport, and does not delay the window's load event.

↪ 7.mp3
The audio loads and autoplay playback begins only once scrolled into the viewport, and does not delay the window's load event.

↪ 8.mp3
The audio's metadata loads only once scrolled into the viewport, and does not delay the window's load event.

For web developers (non-normative)

***audio* = new Audio[p427]([*url*])**

Returns a new audio[p425] element, with the src[p433] attribute set to the value passed in the argument, if applicable.

A legacy factory function is provided for creating HTMLAudioElement[p426] 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[p1151]'s associated Document[p966].
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[p446]" and "auto[p446]".
4. If *src* is given, then set an attribute value for *audio* using "src[p433]" and *src*. (This will cause the user agent to invoke[p433] the object's resource selection algorithm[p436] before returning.)
5. Return *audio*.

4.8.10 The track element §[p427]

Categories[p154]:
None.

Contexts in which this element can be used[p154]:
As a child of a media element[p430], before any flow content[p157].

Content model[p154]:
Nothing[p155].

Tag omission in text/html[p154]:
No end tag[p1359].

Content attributes[p154]:
Global attributes[p162]
kind[p428] — The type of text track
src[p428] — Address of the resource
srclang[p428] — Language of the text track
label[p429] — User-visible label
default[p429] — Enable the track if no other text track[p466] is more suitable

Accessibility considerations[p154]:
For authors.
For implementers.

Sanitization[p154]**:**

Uncategorized[p1249].

DOM interface[p154]**:**

IDL

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

  [CEReactions] attribute DOMString kind;
  [CEReactions, ReflectURL] attribute USVString src;
  [CEReactions, Reflect] attribute DOMString srclang;
  [CEReactions, Reflect] attribute DOMString label;
  [CEReactions, Reflect] 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[p427] element allows authors to specify explicit external timed text tracks[p466] for media elements[p430]. It does not represent[p148] anything on its own.

The **kind** attribute is an enumerated attribute[p80] with the following keywords and states:

Keyword	State	Brief description
subtitles	**Subtitles**	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[p432]'s audio track). Overlaid on the video.
captions	**Captions**	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.
descriptions	**Descriptions**	Textual descriptions of the video component of the media resource[p432], intended for audio synthesis when the visual component is obscured, unavailable, or not usable (e.g. because the user is interacting with the application without a screen while driving, or because the user is blind). Synthesized as audio.
chapters	**Chapters metadata**	Tracks intended for use from script. Not displayed by the user agent.
metadata	**Metadata**	

The attribute's *missing value default*[p80] is the subtitles[p428] state, and its *invalid value default*[p80] is the metadata[p428] 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[p100]. This attribute must be present.

The element has an associated **track URL** (a string), initially the empty string.

When the element's src[p428] attribute is set, run these steps:

1. Let *trackURL* be failure.
2. Let *value* be the element's src[p428] attribute value.
3. If *value* is not the empty string, then set *trackURL* to the result of encoding-parsing-and-serializing a URL[p101] given *value*, relative to the element's node document.
4. Set the element's track URL[p428] to *trackURL* if it is not failure; otherwise to the empty string.

If the element's track URL[p428] identifies a WebVTT resource, and the element's kind[p428] attribute is not in the chapters metadata[p428] or metadata[p428] state, then the WebVTT file must be a WebVTT file using caption or subtitle cue text. [WEBVTT][p1588]

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[p428] attribute is in the subtitles[p428] state. [BCP47][p1579]

If the element has a srclang[p428] 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[p429].

The **label** attribute gives a user-readable title for the track. This title is used by user agents when listing subtitle[p428], caption[p428], and audio description[p428] tracks in their user interface.

The value of the label[p429] attribute, if the attribute is present, must not be the empty string. Furthermore, there must not be two track[p427] element children of the same media element[p430] whose kind[p428] attributes are in the same state, whose srclang[p428] attributes are both missing or have values that represent the same language, and whose label[p429] attributes are again both missing or both have the same value.

If the element has a label[p429] 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[p429] is an empty string.

The **default** attribute is a boolean attribute[p79], 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[p430] must have no more than one track[p427] element child whose kind[p428] attribute is in the subtitles[p428] or captions[p428] state and whose default[p429] attribute is specified.

Each media element[p430] must have no more than one track[p427] element child whose kind[p428] attribute is in the description[p428] state and whose default[p429] attribute is specified.

Each media element[p430] must have no more than one track[p427] element child whose kind[p428] attribute is in the chapters metadata[p428] state and whose default[p429] attribute is specified.

Note

There is no limit on the number of track[p427] elements whose kind[p428] attribute is in the metadata[p428] state and whose default[p429] attribute is specified.

For web developers (non-normative)

***track*.readyState**[p429]

Returns the text track readiness state[p467], represented by a number from the following list:

***track*.NONE**[p429] **(0)**

The text track not loaded[p467] state.

***track*.LOADING**[p429] **(1)**

The text track loading[p467] state.

***track*.LOADED**[p429] **(2)**

The text track loaded[p467] state.

***track*.ERROR**[p429] **(3)**

The text track failed to load[p467] state.

***track*.track**[p430]

Returns the track[p427] element's text track[p466].

The **readyState** attribute must return the numeric value corresponding to the text track readiness state[p467] of the track[p427] element's text track[p466], as defined by the following list:

NONE (numeric value 0)

The text track not loaded[p467] state.

LOADING (numeric value 1)

The text track loading[p467] state.

LOADED (numeric value 2)

The text track loaded[p467] state.

ERROR (numeric value 3)

The text track failed to load[p467] state.

The **track** getter steps are to return this's text track[p466].

The **kind** IDL attribute must reflect[p108] the content attribute of the same name, limited to only known values[p109].

Example

This video has subtitles in several languages:

```
<video src="brave.webm">
 <track kind=subtitles src=brave.en.vtt srclang=en label="English">
 <track kind=captions src=brave.en.hoh.vtt srclang=en label="English for the Hard of Hearing">
 <track kind=subtitles src=brave.fr.vtt srclang=fr lang=fr label="Français">
 <track kind=subtitles src=brave.de.vtt srclang=de lang=de label="Deutsch">
</video>
```

(The lang[p165] attributes on the last two describe the language of the label[p429] attribute, not the language of the subtitles themselves. The language of the subtitles is given by the srclang[p428] attribute.)

4.8.11 Media elements §[p430]

HTMLMediaElement[p430] objects (audio[p425] and video[p420], 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, ReflectURL] 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;

  // playback state
  attribute double currentTime;
  undefined fastSeek(double time);
  readonly attribute unrestricted double duration;
```

```
  object getStartDate();
  readonly attribute boolean paused;
  attribute double defaultPlaybackRate;
  attribute double playbackRate;
  attribute boolean preservesPitch;
  readonly attribute TimeRanges played;
  readonly attribute TimeRanges seekable;
  readonly attribute boolean ended;
  [CEReactions, Reflect] attribute boolean autoplay;
  [CEReactions, Reflect] attribute boolean loop;
  Promise<undefined> play();
  undefined pause();

  // controls
  [CEReactions, Reflect] attribute boolean controls;
  attribute double volume;
  attribute boolean muted;
  [CEReactions, Reflect="muted"] attribute boolean defaultMuted;
  [CEReactions] attribute DOMString loading;

  // tracks
  [SameObject] readonly attribute AudioTrackList audioTracks;
  [SameObject] readonly attribute VideoTrackList videoTracks;
  [SameObject] readonly attribute TextTrackList textTracks;
  TextTrack addTextTrack(TextTrackKind kind, optional DOMString label = "", optional DOMString language
= "");
};
```

The **media element attributes**, src[p433], crossorigin[p433], preload[p446], autoplay[p452], loop[p450], muted[p483], controls[p481], and loading[p431], apply to all media elements[p430]. They are defined in this section.

The **loading** attribute is a lazy loading attribute[p105]. Its purpose is to indicate the policy for loading media resources that are outside the viewport.

When the loading[p431] attribute's state is changed to the Eager[p105] state, the user agent must run these steps:

1. Let *resumptionSteps* be the media element[p430]'s lazy load resumption steps[p105].
2. Let *posterResumptionSteps* be null.
3. If the media element[p430] is a video[p420] element, then set *posterResumptionSteps* to the video[p420] element's poster lazy load resumption steps[p105].
4. If *resumptionSteps* is null and *posterResumptionSteps* is null, then return.
5. If *resumptionSteps* is not null:
    1. Set the media element[p430]'s lazy load resumption steps[p105] to null.
    2. Invoke *resumptionSteps*.
6. If *posterResumptionSteps* is not null:
    1. Set the video[p420] element's poster lazy load resumption steps[p105] to null.
    2. Invoke *posterResumptionSteps*.

The **loading** IDL attribute must reflect[p108] the loading[p431] content attribute, limited to only known values[p109].

When the loading[p431] attribute is in the Lazy[p105] state, it takes precedence over the preload[p446] attribute by deferring data fetching.

If the autoplay[p452] attribute is present and the loading[p431] attribute is in the Lazy[p105] state, the user agent must also defer starting playback (and any associated network requests autoplay can introduce) until the element's lazy load resumption steps[p105] are invoked.

Media elements[p430] 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[p430] 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[p432] has an associated **origin**, which is either "`none`", "`multiple`", "`rewritten`", or an origin[p939]. It is initially set to "`none`".

A media resource[p432] can have multiple audio and video tracks. For the purposes of a media element[p430], the video data of the media resource[p432] is only that of the currently selected track (if any) as given by the element's `videoTracks`[p462] attribute when the event loop[p1193] last reached step 1[p1196], and the audio data of the media resource[p432] is the result of mixing all the currently enabled tracks (if any) given by the element's `audioTracks`[p462] attribute when the event loop[p1193] last reached step 1[p1196].

Note

Both `audio`[p425] and `video`[p420] elements can be used for both audio and video. The main difference between the two is simply that the `audio`[p425] element has no playback area for visual content (such as video or captions), whereas the `video`[p420] element does.

Each media element[p430] has a unique **media element event task source**.

To **queue a media element task** with a media element[p430] *element* and a series of steps *steps*, queue an element task[p1195] on the media element[p430]'s media element event task source[p432] given *element* and *steps*.

✔ MDN

4.8.11.1 Error codes §[p432]

For web developers (non-normative)

***media*.error**[p432]

Returns a `MediaError`[p432] object representing the current error state of the element.

Returns null if there is no error.

All media elements[p430] have an associated error status, which records the last error the element encountered since its resource selection algorithm[p436] was last invoked. The **`error`** attribute, on getting, must return the `MediaError`[p432] object created for this last error, or null if there has not been an error.

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**[p432]**.code**[p433]

Returns the current error's error code, from the list below.

***media*.error**[p432]**.message**[p433]

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`[p432] 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[p432] 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[p432], 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[p432], after the resource was established to be usable.

`MEDIA_ERR_SRC_NOT_SUPPORTED` (numeric value 4)

The media resource[p432] indicated by the `src`[p433] attribute or assigned media provider object[p433] was not suitable.

To **create a `MediaError`**, given an error code which is one of the above values, return a new `MediaError`[p432] object whose code[p432] is the given error code and whose message[p432] 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[p432] must be set to the empty string.

The **`code`** getter steps are to return this's code[p432].

The **`message`** getter steps are to return this's message[p432].

4.8.11.2 Location of the media resource §[p433]

The **`src`** content attribute on media elements[p430] 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[p100].

If the `itemprop`[p833] attribute is specified on the media element[p430], then the `src`[p433] attribute must also be specified.

The **`crossorigin`** content attribute on media elements[p430] is a CORS settings attribute[p103].

If a media element[p430] is created with a `src`[p433] attribute, the user agent must immediately[p44] invoke the media element[p430]'s resource selection algorithm[p436].

If a `src`[p433] attribute of a media element[p430] is set or changed, the user agent must invoke the media element[p430]'s media element load algorithm[p435]. (*Removing* the `src`[p433] attribute does not do this, even if there are `source`[p355] elements present.)

The **`crossOrigin`** IDL attribute must reflect[p108] the `crossorigin`[p433] content attribute, limited to only known values[p109].

✓ MDN

A **media provider object** is an object that can represent a media resource[p432], separate from a URL. `MediaStream` objects, `MediaSource` objects, and `Blob` objects are all media provider objects[p433].

Each media element[p430] has an **assigned media provider object**, which is a media provider object[p433] or null, initially null.

For web developers (non-normative)

***media*.srcObject**[p433] **[= *source*]**

Allows the media element[p430] to be assigned a media provider object[p433].

***media*.currentSrc**[p433]

Returns the URL of the current media resource[p432], if any.

Returns the empty string when there is no media resource[p432], or it doesn't have a URL.

The **`currentSrc`** IDL attribute must initially be set to the empty string. Its value is changed by the resource selection algorithm[p436] defined below.

The **`srcObject`** getter steps are to return this's assigned media provider object[p433].

The `srcObject`[p433] setter steps are:

1. Set this's assigned media provider object[p433] to the given value.
2. Invoke this's media element load algorithm[p435].

Note

There are three ways to specify a media resource[p432]: the `srcObject`[p433] IDL attribute, the `src`[p433] content attribute, and `source`[p355] elements. The IDL attribute takes priority, followed by the content attribute, followed by the elements.

4.8.11.3 MIME types §[p434]

A media resource[p432] can be described in terms of its *type*, specifically a MIME type, in some cases with a `codecs` parameter. (Whether the `codecs` parameter is allowed or not depends on the MIME type.) [RFC6381][p1586]

Types are usually somewhat incomplete descriptions; for example "`video/mpeg`" doesn't say anything except what the container type is, and even a type like "`video/mp4; codecs="avc1.42E01E, mp4a.40.2"`" doesn't include information like the actual bitrate (only the maximum bitrate). Thus, given a type, a user agent can often only know whether it *might* be able to play media of that type (with varying levels of confidence), or whether it definitely *cannot* play media of that type.

A type that the user agent knows it cannot render is one that describes a resource that the user agent definitely does not support, for example because it doesn't recognize the container type, or it doesn't support the listed codecs.

The MIME type "`application/octet-stream`" with no parameters is never a type that the user agent knows it cannot render[p434]. User agents must treat that type as equivalent to the lack of any explicit Content-Type metadata[p102] when it is used to label a potential media resource[p432].

Note

Only the MIME type "`application/octet-stream`" with no parameters is special-cased here; if any parameter appears with it, it will be treated just like any other MIME type. This is a deviation from the rule that unknown MIME type parameters should be ignored.

For web developers (non-normative)

***media*.canPlayType[p434](*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[p434] or is the type "`application/octet-stream`"; it must return "**`probably`**" if the user agent is confident that the type represents a media resource[p432] that it can render if used with this `audio`[p425] or `video`[p420] element; and it must return "**`maybe`**" otherwise. Implementers are encouraged to return "`maybe`[p434]" unless the type can be confidently established as being supported or not. Generally, a user agent should never return "`probably`[p434]" 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`[p420] element:

```
<section id="video">
 <p><a href="playing-cats.nfv">Download video</a></p>
</section>
<script>
 const videoSection = document.getElementById('video');
 const videoElement = document.createElement('video');
 const support = videoElement.canPlayType('video/x-new-fictional-format;codecs="kittens,bunnies"');
 if (support === "probably") {
   videoElement.setAttribute("src", "playing-cats.nfv");
   videoSection.replaceChildren(videoElement);
 }
</script>
```

Note

The `type`[p356] attribute of the `source`[p355] element allows the user agent to avoid downloading resources that use formats it cannot render.

4.8.11.4 Network states §[p434]

For web developers (non-normative)

***media*.networkState**[p435]

Returns the current state of network activity for the element, from the codes in the list below.

As media elements[p430] 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[p436] is active and has selected a resource[p432], but it is not actually using the network at this time.

NETWORK_LOADING (numeric value 2)

The user agent is actively trying to download data.

NETWORK_NO_SOURCE (numeric value 3)

The element's resource selection algorithm[p436] is active, but it has not yet found a resource[p432] to use.

The resource selection algorithm[p436] defined below describes exactly when the networkState[p435] attribute changes value and what events fire to indicate changes in this state.

4.8.11.5 Loading the media resource §[p435]

For web developers (non-normative)

***media*.load()**[p435]

Causes the element to reset and start selecting and loading a new media resource[p432] from scratch.

All media elements[p430] 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[p435] is true, the element must delay the load event[p1458] of its document.

When the **load()** method on a media element[p430] is invoked, the user agent must run the following steps:

1. Let *resumptionSteps* be the media element[p430]'s lazy load resumption steps[p105].
2. If *resumptionSteps* is not null:
    1. Set the media element[p430]'s lazy load resumption steps[p105] to null.
    2. Invoke *resumptionSteps*.
3. Run the media element load algorithm[p435].

A media element[p430] has an associated boolean **is currently stalled**, which is initially false.

The **media element load algorithm** consists of the following steps:

1. Set this element's is currently stalled[p435] to false.
2. Abort any already-running instance of the resource selection algorithm[p436] for this element.
3. Let *pending tasks* be a list of all tasks[p1194] from the media element[p430]'s media element event task source[p432] in one of the task queues[p1193].
4. For each task in *pending tasks* that would resolve pending play promises[p455] or reject pending play promises[p455], immediately resolve or reject those promises in the order the corresponding tasks were queued.
5. Remove each task[p1194] in *pending tasks* from its task queue[p1193].

Note

Basically, pending events and callbacks are discarded and promises in-flight to be resolved/rejected are resolved/

rejected immediately when the media element starts loading a new resource.

6. 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.

7. If the media element's networkState is not set to NETWORK_EMPTY:

    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:

        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 to Not-a-Number (NaN).

    10. Update the duration attribute to Not-a-Number (NaN).

        Note

        The user agent will not fire a durationchange event for this particular change of the duration.

8. Set the playbackRate attribute to the value of the defaultPlaybackRate attribute.

9. Set the error attribute to null and the can autoplay flag to true.

10. Invoke the media element's resource selection algorithm.

Note

11. *Playback of any previously playing media resource for this element stops.*

The **resource selection algorithm** for a media element is as follows. This algorithm is always invoked as part of a task, but one of the first steps in the algorithm is to return and continue running the remaining steps in parallel. In addition, this algorithm interacts closely with the event loop mechanism; in particular, it has synchronous sections (which are triggered as part of the event loop algorithm). Steps in such sections are marked with ⌛.

1. Set the element's networkState attribute to the NETWORK_NO_SOURCE value.

2. Set the element's show poster flag to true.

3. Await a stable state, allowing the task that invoked this algorithm to continue. The synchronous section consists of all the remaining steps of this algorithm until the algorithm says the synchronous section has ended. (Steps in synchronous sections are marked with ⌛.)

4. ⌛ If the media element's lazy loading attribute is in the Eager state or scripting is disabled, then set the media element's delaying-the-load-event flag to true (this delays the load event).

5. ⌛ If the media element's blocked-on-parser flag is false, then populate the list of pending text tracks.

6. ⌛ Let *mode* be null.
7. ⌛ Let *candidate* be null.
8. ⌛ If the media element[p430]'s assigned media provider object[p433] is not null, then set *mode* to *object*.
9. ⌛ Otherwise, if the media element[p430] has a `src`[p433] attribute, then set *mode* to *attribute*.
10. ⌛ Otherwise, if the media element[p430] has a `source`[p355] element child, then set *mode* to *children* and set *candidate* to the first `source`[p355] element child in tree order.
11. ⌛ Otherwise:
    1. ⌛ Set the `networkState`[p435] to `NETWORK_EMPTY`[p435].
    2. ⌛ Set the element's delaying-the-load-event flag[p435] to false. This stops delaying the load event[p1458].
    3. End the synchronous section[p1201] and return.
12. ⌛ Set the media element[p430]'s `networkState`[p435] to `NETWORK_LOADING`[p435].
13. ⌛ Queue a media element task[p432] given the media element[p430] to fire an event named `loadstart`[p484] at the media element[p430].
14. Run the appropriate steps from the following list:

    ↪ **If *mode* is *object***

    1. ⌛ Set the `currentSrc`[p433] attribute to the empty string.
    2. End the synchronous section[p1201], continuing the remaining steps in parallel[p44].
    3. Run the resource fetch algorithm[p440] with the assigned media provider object[p433]. 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[p455] and queue a media element task[p432] given the media element[p430] to run the dedicated media source failure steps[p439] with the result.
    5. Wait for the task[p1194] 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.

    ↪ **If *mode* is *attribute***

    1. ⌛ If the `src`[p433] attribute's value is the empty string, then end the synchronous section[p1201], and jump down to the *failed with attribute* step below.
    2. ⌛ Let *urlRecord* be the result of encoding-parsing a URL[p101] given the `src`[p433] attribute's value, relative to the media element[p430]'s node document when the `src`[p433] attribute was last changed.
    3. ⌛ If *urlRecord* is not failure, then set the `currentSrc`[p433] attribute to the result of applying the URL serializer to *urlRecord*.
    4. End the synchronous section[p1201], continuing the remaining steps in parallel[p44].
    5. If *urlRecord* is not failure, then run the resource fetch algorithm[p440] with *urlRecord*. If that algorithm returns without aborting *this* one, then the load failed.
    6. *Failed with attribute*: Reaching this step indicates that the media resource failed to load or that *urlRecord* is failure. Take pending play promises[p455] and queue a media element task[p432] given the media element[p430] to run the dedicated media source failure steps[p439] with the result.
    7. Wait for the task[p1194] 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.

    ↪ **Otherwise (*mode* is *children*)**

    1. ⌛ Let *pointer* be a position defined by two adjacent nodes in the media element[p430]'s child list, treating the start of the list (before the first child in the list, if any) and end of the list (after the last child in the list, if any)

as nodes in their own right. One node is the node before *pointer*, and the other node is the node after *pointer*. Initially, let *pointer* be the position between the *candidate* node and the next node, if there are any, or the end of the list, if it is the last node.

As nodes are inserted, removed, and moved into the media element[p430], *pointer* must be updated as follows:

If a new node is inserted or moved between the two nodes that define *pointer*
Let *pointer* be the point between the node before *pointer* and the new node. In other words, insertions at *pointer* go after *pointer*.

If the node before *pointer* is removed
Let *pointer* be the point between the node after *pointer* and the node before the node after *pointer*. In other words, *pointer* doesn't move relative to the remaining nodes.

If the node after *pointer* is removed
Let *pointer* be the point between the node before *pointer* and the node after the node before *pointer*. Just as with the previous case, *pointer* doesn't move relative to the remaining nodes.

Other changes don't affect *pointer*.

2. ⌛ *Process candidate*: If *candidate* does not have a `src`[p357] attribute, or if its `src`[p357] attribute's value is the empty string, then end the synchronous section[p1201], and jump down to the *failed with elements* step below.
3. ⌛ If *candidate* has a `media`[p356] attribute whose value does not match the environment[p100], then end the synchronous section[p1201], and jump down to the *failed with elements* step below.
4. ⌛ Let *urlRecord* be the result of encoding-parsing a URL[p101] given *candidate*'s `src`[p433] attribute's value, relative to *candidate*'s node document when the `src`[p433] attribute was last changed.
5. ⌛ If *urlRecord* is failure, then end the synchronous section[p1201], and jump down to the *failed with elements* step below.
6. ⌛ If *candidate* has a `type`[p356] 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[p434], then end the synchronous section[p1201], and jump down to the *failed with elements* step below.
7. ⌛ Set the `currentSrc`[p433] attribute to the result of applying the URL serializer to *urlRecord*.
8. End the synchronous section[p1201], continuing the remaining steps in parallel[p44].
9. Run the resource fetch algorithm[p440] with *urlRecord*. If that algorithm returns without aborting *this* one, then the load failed.
10. *Failed with elements*: Queue a media element task[p432] given the media element[p430] to fire an event named `error`[p486] at *candidate*.
11. Await a stable state[p1201]. The synchronous section[p1201] consists of all the remaining steps of this algorithm until the algorithm says the synchronous section[p1201] has ended. (Steps in synchronous sections[p1201] are marked with ⌛.)
12. ⌛ Forget the media element's media-resource-specific tracks[p446].
13. ⌛ *Find next candidate*: Let *candidate* be null.
14. ⌛ *Search loop*: If the node after *pointer* is the end of the list, then jump to the *waiting* step below.
15. ⌛ If the node after *pointer* is a `source`[p355] element, let *candidate* be that element.
16. ⌛ 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.
17. ⌛ If *candidate* is null, jump back to the *search loop* step. Otherwise, jump back to the *process candidate* step.
18. ⌛ *Waiting*: Set the element's `networkState`[p435] attribute to the `NETWORK_NO_SOURCE`[p435] value.
19. ⌛ Set the element's show poster flag[p449] to true.
20. ⌛ Queue a media element task[p432] given the media element[p430] to set the element's delaying-the-load-event flag[p435] to false. This stops delaying the load event[p1458].

21. End the synchronous section[p1201], continuing the remaining steps in parallel[p44].

22. Wait until the node after *pointer* is a node other than the end of the list. (This step might wait forever.)

23. Await a stable state[p1201]. The synchronous section[p1201] consists of all the remaining steps of this algorithm until the algorithm says the synchronous section[p1201] has ended. (Steps in synchronous sections[p1201] are marked with ⌛.)

24. ⌛ If the media element[p430]'s lazy loading attribute[p105] is in the Eager[p105] state or scripting is disabled[p1152], then set the element's delaying-the-load-event flag[p435] back to true (this delays the load event[p1458] again, in case it hasn't been fired yet).

25. ⌛ Set the `networkState`[p435] back to `NETWORK_LOADING`[p435].

26. ⌛ 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`[p432] attribute to the result of creating a `MediaError`[p433] with `MEDIA_ERR_SRC_NOT_SUPPORTED`[p433].
2. Forget the media element's media-resource-specific tracks[p446].
3. Set the element's `networkState`[p435] attribute to the `NETWORK_NO_SOURCE`[p435] value.
4. Set the element's show poster flag[p449] to true.
5. Fire an event named `error`[p485] at the media element[p430].
6. Reject pending play promises[p455] with *promises* and a `"NotSupportedError"` `DOMException`.
7. Set the element's delaying-the-load-event flag[p435] to false. This stops delaying the load event[p1458].

To **verify a media response** given a response *response*, a media resource[p432] *resource*, and "`entire resource`" or a (number, number or "`until end`") tuple *byteRange*:

1. If *response* is a network error, then return false.
2. If *byteRange* is "`entire resource`", then return true.
3. Let *internalResponse* be *response*'s unsafe response[p102].
4. If *internalResponse*'s status is 200, then return true.
5. If *internalResponse*'s status is not 206, then return false.
6. If the result of extracting content-range values from *internalResponse* is failure, then return false.

   Note

   Note that the extracted values are not used, and in particular are not compared to byteRange*. So this step serves as syntactic validation of the `Content-Range` header, but if the `Content-Range` values on the response mismatch the `Range` values on the request, that is not considered a failure.*

7. Let *origin* be "`rewritten`" if *internalResponse*'s URL is null; otherwise *internalResponse*'s URL's origin.
8. Let *previousOrigin* be *resource*'s origin[p432].
9. If any of the following are true:
   - *previousOrigin* is "`none`";
   - *origin* and *previousOrigin* are "`rewritten`"; or
   - *origin* and *previousOrigin* are origins[p939], and *origin* is same origin[p940] with *previousOrigin*,

   then set *resource*'s origin[p432] to *origin*.

   Otherwise, if *response* is CORS-cross-origin[p102], then return false.

   Otherwise, set *resource*'s origin[p432] to "`multiple`".

Note

This ensures that opaque responses with range headers do not leak information by being patched together with other responses from different origins.

10. Return true.

The **resource fetch algorithm** for a media element[p430] and a given URL record or media provider object[p433] is as follows:

1. If the will lazy load element steps[p105] given the media element[p430] return true:
    1. Let *resumptionSteps* be the rest of this algorithm starting with the step labeled *Let mode be remote*.
    2. Set the media element[p430]'s lazy load resumption steps[p105] to *resumptionSteps*.
    3. Start intersection-observing a lazy loading element[p106] for the media element[p430].
    4. Return.
2. Let *mode* be *remote*.
3. If the algorithm was invoked with media provider object[p433], then set *mode* to *local*.

    Otherwise:
    1. Let *isTopLevelSelfFetch* be false.
    2. Let *settingsObject* be the media element[p430]'s node document's relevant settings object[p1151].
    3. Let *global* be the media element[p430]'s node document's relevant global object[p1151].
    4. If all of the following conditions are true:
        - *global* is a `Window`[p965] object;
        - *global*'s navigable[p966] is not null;
        - *global*'s navigable[p966]'s parent[p1035] is null; and
        - *settingsObject*'s creation URL[p1143] equals the URL record,

        then set *isTopLevelSelfFetch* to true.
    5. Let *stringOrEnvironment* be "`top-level-self-fetch`" if *isTopLevelSelfFetch* is true; otherwise *settingsObject*.
    6. Let *object* be the result of obtaining a blob object using the URL record's blob URL entry and *stringOrEnvironment*.
    7. If *object* is a media provider object[p433], then set *mode* to *local*.
4. If *mode* is *remote*, then let the *current media resource* be the resource given by the URL record passed to this algorithm; otherwise, let the *current media resource* be the resource given by the media provider object[p433]. Either way, the *current media resource* is now the element's media resource[p432].
5. Remove all media-resource-specific text tracks[p469] from the media element[p430]'s list of pending text tracks[p468], if any.
6. Run the appropriate steps from the following list:

    ↪ If *mode* is remote

    1. Optionally, run the following substeps. This is the expected behavior if the user agent intends to not attempt to fetch the resource until the user requests it explicitly (e.g. as a way to implement the `preload`[p446] attribute's `none`[p446] keyword).
        1. Set the `networkState`[p435] to `NETWORK_IDLE`[p435].
        2. Queue a media element task[p432] given the media element[p430] to fire an event named `suspend`[p484] at the element.
        3. Queue a media element task[p432] given the media element[p430] to set the element's delaying-the-load-event flag[p435] to false. This stops delaying the load event[p1458].
        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[p435] back to true (this delays the load event[p1458] again, in case it hasn't been fired yet).

7. Set the `networkState`[p435] to `NETWORK_LOADING`[p435].

2. Let *destination* be "`audio`" if the media element[p430] is an `audio`[p425] element, or "`video`" otherwise.

3. Let *request* be the result of creating a potential-CORS request[p102] given *current media resource*'s URL record, *destination*, and the current state of the media element[p430]'s `crossorigin`[p688] content attribute.

4. Set *request*'s client to the media element[p430]'s node document's relevant settings object[p1151].

5. Set *request*'s initiator type to *destination*.

6. Let *byteRange*, which is "`entire resource`" or a (number, number or "`until end`") tuple, be the byte range required to satisfy missing data in media data[p431]. This value is implementation-defined and may rely on codec, network conditions or other heuristics. The user-agent may determine to fetch the resource in full, in which case *byteRange* would be "`entire resource`", to fetch from a byte offset until the end, in which case *byteRange* would be (number, "`until end`"), or to fetch a range between two byte offsets, in which case *byteRange* would be a (number, number) tuple representing the two offsets.

7. If *byteRange* is not "`entire resource`":

    1. If *byteRange*[1] is "`until end`", then add a range header to *request* given *byteRange*[0].

    2. Otherwise, add a range header to *request* given *byteRange*[0] and *byteRange*[1].

8. Fetch *request*, with *processResponse* set to the following steps given response *response*:

    1. Let *global* be the media element[p430]'s node document's relevant global object[p1151].

    2. Let *updateMedia* be to queue a media element task[p432] given the media element[p430] to run the first appropriate steps from the media data processing steps list[p442] below. (A new task is used for this so that the work described below occurs relative to the appropriate media element event task source[p432] rather than using the networking task source[p1204].)

    3. Let *processEndOfMedia* be the following step: If the fetching process has completed 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.

    4. If the result of verifying[p439] *response* given the *current media resource* and *byteRange* is false, then abort these steps.

    5. Otherwise, incrementally read *response*'s body given *updateMedia*, *processEndOfMedia*, an empty algorithm, and *global*.

    6. Update the media data[p431] with the contents of *response*'s unsafe response[p102] obtained in this fashion. *response* can be CORS-same-origin[p102] or CORS-cross-origin[p102]; this affects whether subtitles referenced in the media data[p431] are exposed in the API and, for `video`[p420] elements, whether a `canvas`[p713] gets tainted when the video is drawn on it.

The **media element stall timeout** is an implementation-defined length of time, which should be about three seconds. When a media element[p430] that is actively attempting to obtain media data[p431] has failed to receive any data for a duration equal to the media element stall timeout[p441], the user agent must queue a media element task[p432] given the media element[p430] to:

1. Set the element's is currently stalled[p435] to true.

2. Fire an event named `stalled`[p485] at the element.

User agents may allow users to selectively block or slow media data[p431] downloads. When a media element[p430]'s download has been blocked altogether, the user agent must act as if it was stalled (as opposed to acting as if the connection was closed). The rate of the download may also be throttled automatically by the user agent, e.g. to balance the download with other connections sharing the same bandwidth.

User agents may decide to not download more content at any time, e.g. after buffering five minutes of a one hour media resource, while waiting for the user to decide whether to play the resource or not, while waiting for user input in an interactive resource, or when the user navigates away from the page. When a media element[p430]'s download has been suspended, the user agent must queue a media element task[p432] given the media element[p430] to set the `networkState`[p435] to `NETWORK_IDLE`[p435] and fire an event named `suspend`[p484] at the element. If and when downloading of the resource resumes, the user agent must queue a media element task[p432] given the media element[p430] to set the `networkState`[p435] to `NETWORK_LOADING`[p435]. Between the queuing of these tasks, the load is suspended (so `progress`[p484] events don't fire, as described above).

Note

The `preload`[p446] attribute provides a hint regarding how much buffering the author thinks is advisable, even in the absence of the `autoplay`[p452] attribute.

When a user agent decides to completely suspend a download, e.g., if it is waiting until the user starts playback before downloading any further content, the user agent must queue a media element task[p432] given the media element[p430] to set the element's delaying-the-load-event flag[p435] to false. This stops delaying the load event[p1458].

Although the above steps give an algorithm for issuing requests, the user agent may use other means besides those exact ones, especially in the face of error conditions. For example, the user agent may reconnect to the server or switch to a streaming protocol. The user agent must only consider the resource erroneous, and proceed into the error branches of the above steps, if the user agent has given up trying to fetch the resource.

To determine the format of the media resource[p432], 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[p432] given the media element[p430] to:

1. Set the element's is currently stalled[p435] to false.
2. Fire an event named `progress`[p484] at the element.

While the user agent might still need network access to obtain parts of the media resource[p432], 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[p453], because there is always the chance the user will seek back to the start. In fact, in this situation, once playback has ended[p453], the user agent will end up firing a `suspend`[p484] event, as described earlier.

↪ **Otherwise (*mode* is *local*)**

The resource described by the *current media resource*, if any, contains the media data[p431]. It is CORS-same-origin[p102].

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[p432], 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[p432] given the media element[p430] to run the first appropriate steps from the media data processing steps list[p442] 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[p431] 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[p431] 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[p436].

↪ If the media resource[p432] is found to have an audio track

1. Create an AudioTrack[p462] object to represent the audio track.
2. Update the media element[p430]'s audioTracks[p462] attribute's AudioTrackList[p462] object with the new AudioTrack[p462] object.
3. Let *enable* be *unknown*.
4. If either the media resource[p432] 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: 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[p430] does not yet have an enabled[p465] audio track, then set *enable* to *true*, otherwise, set *enable* to *false*.
6. If *enable* is *true*, then enable this audio track, otherwise, do not enable this audio track.
7. Fire an event named addtrack[p486] at this AudioTrackList[p462] object, using TrackEvent[p484], with the track[p484] attribute initialized to the new AudioTrack[p462] object.

↪ If the media resource[p432] is found to have a video track

1. Create a VideoTrack[p462] object to represent the video track.
2. Update the media element[p430]'s videoTracks[p462] attribute's VideoTrackList[p462] object with the new VideoTrack[p462] object.
3. Let *enable* be *unknown*.
4. If either the media resource[p432] 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: 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[p430] does not yet have a selected[p465] 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[p465].
7. Fire an event named addtrack[p486] at this VideoTrackList[p462] object, using TrackEvent[p484], with the track[p484] attribute initialized to the new VideoTrack[p462] object.

↪ Once enough of the media data[p431] has been fetched to determine the duration of the media resource[p432], its dimensions, and other metadata

This indicates that the resource is usable. The user agent must follow these substeps:

1. Establish the media timeline[p447] for the purposes of the current playback position[p448] and the earliest possible position[p449], based on the media data[p431].
2. Update the timeline offset[p449] to the date and time that corresponds to the zero time in the media

timeline[p447] established in the previous step, if any. If no explicit time and date is given by the media resource[p432], the timeline offset[p449] must be set to Not-a-Number (NaN).

3. Set the current playback position[p448] and the official playback position[p448] to the earliest possible position[p449].

4. Update the duration[p449] attribute with the time of the last frame of the resource, if known, on the media timeline[p447] established above. If it is not known (e.g. a stream that is in principle infinite), update the duration[p449] attribute to the value positive Infinity.

   Note

   The user agent will[p449] queue a media element task[p432] given the media element[p430] to fire an event named durationchange[p485] at the element at this point.

5. For video[p420] elements, set the videoWidth[p424] and videoHeight[p424] attributes, and queue a media element task[p432] given the media element[p430] to fire an event named resize[p485] at the media element[p430].

   Note

   Further resize[p485] events will be fired if the dimensions subsequently change.

6. Set the readyState[p452] attribute to HAVE_METADATA[p450].

   Note

   A loadedmetadata[p485] DOM event will be fired[p451] as part of setting the readyState[p452] attribute to a new value.

7. Let *jumped* be false.

8. If the media element[p430]'s default playback start position[p448] is greater than zero, then seek[p460] to that time, and let *jumped* be true.

9. Set the media element[p430]'s default playback start position[p448] to zero.

10. Let the *initial playback position* be 0.

11. If either the media resource[p432] 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[p460] 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[p465] audio track, then enable an audio track. This will cause a change event to be fired[p465].

13. If there is no selected[p465] video track, then select a video track. This will cause a change event to be fired[p465].

Once the readyState[p452] attribute reaches HAVE_CURRENT_DATA[p450], after the loadeddata event has been fired[p451], set the element's delaying-the-load-event flag[p435] to false. This stops delaying the load event[p1458].

Note

A user agent that is attempting to reduce network usage while still fetching the metadata for each media resource[p432] would also stop buffering at this point, following the rules described previously[p442], which involve the networkState[p435] attribute switching to the NETWORK_IDLE[p435] value and a suspend[p484] event firing.

Note

The user agent is required *to determine the duration of the media resource[p432] and go through this step before playing.*

↪ **Once the entire media resource[p432] has been fetched (but potentially before any of it has been decoded)**

Fire an event named progress[p484] at the media element[p430].

Set the networkState[p435] to NETWORK_IDLE[p435] and fire an event named suspend[p484] at the media element[p430].

If the user agent ever discards any media data[p431] and then needs to resume the network activity to obtain it again, then it must queue a media element task[p432] given the media element[p430] to set the networkState[p435] to NETWORK_LOADING[p435].

Note

If the user agent can keep the media resource[p432] loaded, then the algorithm will continue to its final step *below, which aborts the algorithm.*

↪ **If the connection is interrupted after some media data[p431] 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[p430]'s readyState[p452] attribute is no longer HAVE_NOTHING[p450]) must cause the user agent to execute the following steps:

1. The user agent should cancel the fetching process.
2. Set the error[p432] attribute to the result of creating a MediaError[p433] with MEDIA_ERR_NETWORK[p432].
3. Set the element's networkState[p435] attribute to the NETWORK_IDLE[p435] value.
4. Set the element's delaying-the-load-event flag[p435] to false. This stops delaying the load event[p1458].
5. Fire an event named error[p485] at the media element[p430].
6. Abort the overall resource selection algorithm[p436].

↪ **If the media data[p431] is corrupted**

Fatal errors in decoding the media data[p431] that occur after the user agent has established whether the *current media resource* is usable (i.e. once the media element[p430]'s readyState[p452] attribute is no longer HAVE_NOTHING[p450]) must cause the user agent to execute the following steps:

1. The user agent should cancel the fetching process.
2. Set the error[p432] attribute to the result of creating a MediaError[p433] with MEDIA_ERR_DECODE[p433].
3. Set the element's networkState[p435] attribute to the NETWORK_IDLE[p435] value.
4. Set the element's delaying-the-load-event flag[p435] to false. This stops delaying the load event[p1458].
5. Fire an event named error[p485] at the media element[p430].
6. Abort the overall resource selection algorithm[p436].

↪ **If the media data[p431] 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()[p435] 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[p432] attribute to the result of creating a MediaError[p433] with MEDIA_ERR_ABORTED[p432].
3. Fire an event named abort[p484] at the media element[p430].
4. If the media element[p430]'s readyState[p452] attribute has a value equal to HAVE_NOTHING[p450], set the element's networkState[p435] attribute to the NETWORK_EMPTY[p435] value, set the element's show poster flag[p449] to true, and fire an event named emptied[p485] at the element.

   Otherwise, set the element's networkState[p435] attribute to the NETWORK_IDLE[p435] value.
5. Set the element's delaying-the-load-event flag[p435] to false. This stops delaying the load event[p1458].
6. Abort the overall resource selection algorithm[p436].

↪ **If the media data[p431] 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[p432] is found to declare a media-resource-specific text track[p469] that the user agent supports**

If the media data[p431] is CORS-same-origin[p102], run the steps to expose a media-resource-specific text track[p469] 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.

7. *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[p436].

When a media element[p430] is to **forget the media element's media-resource-specific tracks**, the user agent must remove from the media element[p430]'s list of text tracks[p466] all the media-resource-specific text tracks[p469], then empty the media element[p430]'s `audioTracks`[p462] attribute's `AudioTrackList`[p462] object, then empty the media element[p430]'s `videoTracks`[p462] attribute's `VideoTrackList`[p462] object. No events (in particular, no `removetrack`[p486] events) are fired as part of this; the `error`[p485] and `emptied`[p485] events, fired by the algorithms that invoke this one, can be used instead.

The **`preload`** attribute is an enumerated attribute[p80] with the following keywords and states:

Keyword	State	Brief description
`auto`	**Automatic**	Hints to the user agent that the user agent can put the user's needs first without risk to the server, up to and including optimistically downloading the entire resource.
`none`	**None**	Hints to the user agent that either the author does not expect the user to need the media resource, or that the server wants to minimize unnecessary traffic. This state does not provide a hint regarding how aggressively to actually download the media resource if buffering starts anyway (e.g. once the user hits "play").
`metadata`	**Metadata**	Hints to the user agent that the author does not expect the user to need the media resource, but that fetching the resource metadata (dimensions, track list, duration, etc.), and maybe even the first few frames, is reasonable. If the user agent precisely fetches no more than the metadata, then the media element[p430] will end up with its `readyState`[p452] attribute set to `HAVE_METADATA`[p450]; typically though, some frames will be obtained as well and it will probably be `HAVE_CURRENT_DATA`[p450] or `HAVE_FUTURE_DATA`[p450]. 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.

The attribute's *empty value default*[p80] is the Automatic[p446] state.

The attribute's *missing value default*[p80] and *invalid value default*[p80] are both implementation-defined, though the Metadata[p446] state is suggested as a compromise between reducing server load and providing an optimal user experience.

The attribute can be changed even once the media resource[p432] is being buffered or played; the descriptions in the table above are to be interpreted with that in mind.

Note

Authors might switch the attribute from "`none`[p446]" or "`metadata`[p446]" to "`auto`[p446]" 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`[p446] 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[p108] the content attribute of the same name, limited to only known values[p109].

Note

The `autoplay`[p452] attribute can override the `preload`[p446] attribute (since if the media plays, it naturally has to buffer first, regardless of the hint given by the `preload`[p446] attribute). Including both is not an error, however.

Note

In audio[p425] and video[p420] elements, if scripting is enabled[p1152], the loading[p431] attribute can defer the preload[p446] attribute's hinted behavior until the element's lazy load resumption steps[p105] are run.

For web developers (non-normative)

***media*.buffered[p447]**

Returns a TimeRanges[p483] object that represents the ranges of the media resource[p432] that the user agent has buffered.

The **buffered** getter steps are to return a new normalized TimeRanges object[p484] that represents the ranges of this's media resource[p432], if any, that the user agent has buffered. User agents must accurately determine the ranges available, even for media streams where this can only be determined by tedious inspection.

Note

Typically this will be a single range anchored at the zero point, but if, e.g. the user agent uses HTTP range requests in response to seeking, then there could be multiple ranges.

User agents may discard previously buffered data.

Note

Thus, a time position included within a range of the objects return by the buffered[p447] attribute at one time can end up being not included in the range(s) of objects returned by the same attribute at later times.

⚠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.

4.8.11.6 Offsets into the media resource §[p447]

For web developers (non-normative)

***media*.duration[p449]**

Returns the length of the media resource[p432], in seconds, assuming that the start of the media resource[p432] is at time zero.

Returns NaN if the duration isn't available.

Returns Infinity for unbounded streams.

***media*.currentTime[p449] [= *value*]**

Returns the official playback position[p448], in seconds.

Can be set, to seek to the given time.

A media resource[p432] has a **media timeline** that maps times (in seconds) to positions in the media resource[p432]. 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[p432] 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[p447] should be that timeline. (Whether the media resource[p432] can specify a timeline or not depends on the media resource[p432]'s format.) If the media resource[p432] specifies an explicit start time *and date*, then that time and date should be considered the zero point in the media timeline[p447]; the timeline offset[p449] will be the time and date, exposed using the getStartDate()[p450] method.

If the media resource[p432] 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[p447] of the media resource[p432] increases linearly starting from the earliest possible position[p449] (as defined below), even if the underlying media data[p431] 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[p432] that does not have an explicit timeline, the zero time on the media timeline[p447] should correspond to the first frame of the media resource[p432]. In the even rarer case of a media resource[p432] 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`[p1547] 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[p432]; otherwise, it should correspond to the first frame received from the server (the point in the media resource[p432] 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()`[p450] 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[p447] starting at 0s and extending to 3,600s (one hour). Assuming the streaming server disconnected at the end of the second clip, the `duration`[p449] attribute would then return 3,600. The `getStartDate()`[p450] 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[p447] starting at 0s and extending to 3,300s (fifty five minutes). In this case, the `getStartDate()`[p450] 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`[p461] 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[p449] (as defined below) using the established media timeline[p447], is greater than or equal to zero.

The media timeline[p447] also has an associated clock. Which clock is used is user-agent defined, and may be media resource[p432]-dependent, but it should approximate the user's wall clock.

Media elements[p430] have a **current playback position**, which must initially (i.e. in the absence of media data[p431]) be zero seconds. The current playback position[p448] is a time on the media timeline[p447].

Media elements[p430] also have an **official playback position**, which must initially be set to zero seconds. The official playback position[p448] is an approximation of the current playback position[p448] that is kept stable while scripts are running.

Media elements[p430] 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[p430] has a **show poster flag**. When a media element[p430] 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`[p420] element instead of showing the video contents.

The **`currentTime`** attribute must, on getting, return the media element[p430]'s default playback start position[p448], unless that is zero, in which case it must return the element's official playback position[p448]. The returned value must be expressed in seconds. On setting, if the media element[p430]'s `readyState`[p452] is `HAVE_NOTHING`[p450], then it must set the media element[p430]'s default playback start position[p448] to the new value; otherwise, it must set the official playback position[p448] to the new value and then seek[p460] to the new value. The new value must be interpreted as being in seconds.

If the media resource[p432] 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[p432] might have a media timeline[p447] 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[p447].

Note

The earliest possible position[p449] is not explicitly exposed in the API; it corresponds to the start time of the first range in the `seekable`[p461] attribute's `TimeRanges`[p483] object, if any, or the current playback position[p448] otherwise.

When the earliest possible position[p449] changes: if the current playback position[p448] is before the earliest possible position[p449], the user agent must seek[p460] to the earliest possible position[p449]; otherwise, if the user agent has not fired a `timeupdate`[p485] 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[p432] given the media element[p430] to fire an event named `timeupdate`[p485] at the element.

Note

Because of the above requirement and the requirement in the resource fetch algorithm[p440] that kicks in when the metadata of the clip becomes known[p443], the current playback position[p448] can never be less than the earliest possible position[p449].

If at any time the user agent learns that an audio or video track has ended and all media data[p431] relating to that track corresponds to parts of the media timeline[p447] that are *before* the earliest possible position[p449], the user agent may queue a media element task[p432] given the media element[p430] to run these steps:

1. Remove the track from the `audioTracks`[p462] attribute's `AudioTrackList`[p462] object or the `videoTracks`[p462] attribute's `VideoTrackList`[p462] object as appropriate.

2. Fire an event named `removetrack`[p486] at the media element[p430]'s aforementioned `AudioTrackList`[p462] or `VideoTrackList`[p462] object, using `TrackEvent`[p484], with the `track`[p484] attribute initialized to the `AudioTrack`[p462] or `VideoTrack`[p462] object representing the track.

The **`duration`** attribute must return the time of the end of the media resource[p432], in seconds, on the media timeline[p447]. If no media data[p431] is available, then the attributes must return the Not-a-Number (NaN) value. If the media resource[p432] 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[p432] before playing any part of the media data[p431] and before setting `readyState`[p452] to a value greater than or equal to `HAVE_METADATA`[p450], even if doing so requires fetching multiple parts of the resource.

When the length of the media resource[p432] 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[p432] given the media element[p430] to fire an event named `durationchange`[p485] at the media element[p430]. (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[p448] ends up being greater than the time of the end of the media resource[p432], then the user agent must also seek[p460] to the time of the end of the media resource[p432].

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`[p485] event would be fired. Similarly, if the user agent initially estimated the media resource[p432]'s duration instead of determining it precisely, and later revises the estimate based on new information, then the duration would change and the `durationchange`[p485] event would be fired.

Some video files also have an explicit date and time corresponding to the zero time in the media timeline[p447], known as the **timeline offset**. Initially, the timeline offset[p449] must be set to Not-a-Number (NaN).

The **getStartDate()** method must return a new Date object[p58] representing the current timeline offset[p449].

The **loop** attribute is a boolean attribute[p79] that, if specified, indicates that the media element[p430] is to seek back to the start of the media resource[p432] upon reaching the end.

4.8.11.7 Ready states §[p450]

For web developers (non-normative)

***media*.readyState**[p452]

Returns a value that expresses the current state of the element with respect to rendering the current playback position[p448], from the codes in the list below.

Media elements[p430] have a *ready state*, which describes to what degree they are ready to be rendered at the current playback position[p448]. 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[p432] is available. No data for the current playback position[p448] is available. Media elements[p430] whose networkState[p435] attribute are set to NETWORK_EMPTY[p435] are always in the HAVE_NOTHING[p450] 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[p420] element, the dimensions of the video are also available. No media data[p431] is available for the immediate current playback position[p448].

HAVE_CURRENT_DATA (numeric value 2)

Data for the immediate current playback position[p448] is available, but either not enough data is available that the user agent could successfully advance the current playback position[p448] in the direction of playback[p457] at all without immediately reverting to the HAVE_METADATA[p450] state, or there is no more data to obtain in the direction of playback[p457]. 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[p448] is at the end of the current frame; and to when playback has ended[p453].

HAVE_FUTURE_DATA (numeric value 3)

Data for the immediate current playback position[p448] is available, as well as enough data for the user agent to advance the current playback position[p448] in the direction of playback[p457] at least a little without immediately reverting to the HAVE_METADATA[p450] state, and the text tracks are ready[p468]. 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[p448] 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[p448] is in the middle of a frame. The user agent cannot be in this state if playback has ended[p453], as the current playback position[p448] can never advance in this case.

HAVE_ENOUGH_DATA (numeric value 4)

All the conditions described for the HAVE_FUTURE_DATA[p450] 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[p448], if it were to advance at the element's playbackRate[p454], would not overtake the available data before playback reaches the end of the media resource[p432].
- 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[p450] and HAVE_CURRENT_DATA[p450] is negligible. Really the only time the difference is relevant is when painting a video[p420] element onto a canvas[p713], where it distinguishes the case where something will be drawn (HAVE_CURRENT_DATA[p450] or greater) from the case where nothing is drawn (HAVE_METADATA[p450] or less). Similarly, the difference between HAVE_CURRENT_DATA[p450] (only the current frame) and HAVE_FUTURE_DATA[p450] (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[p430] whose networkState[p435] is not NETWORK_EMPTY[p435] 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[p450], and the new ready state is HAVE_METADATA[p450]**

   Queue a media element task[p432] given the media element[p430] to fire an event named loadedmetadata[p485] at the element.

   Note

   Before this task is run, as part of the event loop[p1193] mechanism, the rendering will have been updated to resize the video[p420] element if appropriate.

   ↪ **If the previous ready state was HAVE_METADATA[p450] and the new ready state is HAVE_CURRENT_DATA[p450] or greater**

   If this is the first time this occurs for this media element[p430] since the load()[p435] algorithm was last invoked, the user agent must queue a media element task[p432] given the media element[p430] to fire an event named loadeddata[p485] at the element.

   If the new ready state is HAVE_FUTURE_DATA[p450] or HAVE_ENOUGH_DATA[p450], then the relevant steps below must then be run also.

   ↪ **If the previous ready state was HAVE_FUTURE_DATA[p450] or more, and the new ready state is HAVE_CURRENT_DATA[p450] or less**

   If the media element[p430] was potentially playing[p453] before its readyState[p452] attribute changed to a value lower than HAVE_FUTURE_DATA[p450], and the element has not ended playback[p453], and playback has not stopped due to errors[p453], paused for user interaction[p454], or paused for in-band content[p454], the user agent must queue a media element task[p432] given the media element[p430] to fire an event named timeupdate[p485] at the element, and queue a media element task[p432] given the media element[p430] to fire an event named waiting[p485] at the element.

   ↪ **If the previous ready state was HAVE_CURRENT_DATA[p450] or less, and the new ready state is HAVE_FUTURE_DATA[p450]**

   The user agent must queue a media element task[p432] given the media element[p430] to fire an event named canplay[p485] at the element.

   If the element's paused[p453] attribute is false, the user agent must notify about playing[p455] for the element.

   ↪ **If the new ready state is HAVE_ENOUGH_DATA[p450]**

   If the previous ready state was HAVE_CURRENT_DATA[p450] or less, the user agent must queue a media element task[p432] given the media element[p430] to fire an event named canplay[p485] at the element, and, if the element's paused[p453] attribute is false, notify about playing[p455] for the element.

   The user agent must queue a media element task[p432] given the media element[p430] to fire an event named canplaythrough[p485] at the element.

   If the element is not eligible for autoplay[p453], then the user agent must abort these substeps.

   The user agent may run the following substeps:

   1. Set the paused[p453] attribute to false.
   2. If the element's show poster flag[p449] is true, set it to false and run the *time marches on[p458]* steps.
   3. Queue a media element task[p432] given the element to fire an event named play[p485] at the element.
   4. Notify about playing[p455] for the element.

   Alternatively, if the element is a video[p420] element, the user agent may start observing whether the element intersects the viewport[p1488]. When the element starts intersecting the viewport[p1488], if the element is still eligible for autoplay[p453], run the substeps above. Optionally, when the element stops intersecting the viewport[p1488], if the can autoplay flag[p435] is still true and the autoplay[p452] attribute is still specified, run the following substeps:

   1. Run the internal pause steps[p456] and set the can autoplay flag[p435] to true.
   2. Queue a media element task[p432] given the element to fire an event named pause[p485] at the element.

Note

The substeps for playing and pausing can run multiple times as the element starts or stops intersecting the viewport[p1488], as long as the can autoplay flag[p435] is true.

Note

User agents do not need to support autoplay, and it is suggested that user agents honor user preferences on the matter. Authors are urged to use the `autoplay`[p452] attribute rather than using script to force the video to play, so as to allow the user to override the behavior if so desired.

Note

It is possible for the ready state of a media element to jump between these states discontinuously. For example, the state of a media element can jump straight from `HAVE_METADATA`[p450] to `HAVE_ENOUGH_DATA`[p450] without passing through the `HAVE_CURRENT_DATA`[p450] and `HAVE_FUTURE_DATA`[p450] states.

The **`readyState`** IDL attribute must, on getting, return the value described above that describes the current ready state of the media element[p430].

The **`autoplay`** attribute is a boolean attribute[p79]. When present, the user agent (as described in the algorithm described herein) will automatically begin playback of the media resource[p432] as soon as it can do so without stopping.

Note

Authors are urged to use the `autoplay`[p452] attribute rather than using script to trigger automatic playback, as this allows the user to override the automatic playback when it is not desired, e.g. when using a screen reader. Authors are also encouraged to consider not using the automatic playback behavior at all, and instead to let the user agent wait for the user to start playback explicitly.

4.8.11.8 Playing the media resource §[p452]

For web developers (non-normative)

***media*.paused**[p453]

Returns true if playback is paused; false otherwise.

***media*.ended**[p453]

Returns true if playback has reached the end of the media resource[p432].

***media*.defaultPlaybackRate**[p454] **[= *value*]**

Returns the default rate of playback, for when the user is not fast-forwarding or reversing through the media resource[p432].

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**[p454] **[= *value*]**

Returns the current rate playback, where 1.0 is normal speed.

Can be set, to change the rate of playback.

***media*.preservesPitch**[p455]

Returns true if pitch-preserving algorithms are used when the `playbackRate`[p454] is not 1.0. The default value is true.

Can be set to false to have the media resource[p432]'s audio pitch change up or down depending on the `playbackRate`[p454]. This is useful for aesthetic and performance reasons.

***media*.played**[p455]

Returns a `TimeRanges`[p483] object that represents the ranges of the media resource[p432] that the user agent has played.

***media*.play**[p455]**()**

Sets the `paused`[p453] attribute to false, loading the media resource[p432] and beginning playback if necessary. If the playback had ended, will restart it from the start.

***media*.pause[p456]()**

Sets the paused[p453] attribute to true, loading the media resource[p432] if necessary.

The **paused** attribute represents whether the media element[p430] is paused or not. The attribute must initially be true.

A media element[p430] is a **blocked media element** if its readyState[p452] attribute is in the HAVE_NOTHING[p450] state, the HAVE_METADATA[p450] state, or the HAVE_CURRENT_DATA[p450] state, or if the element has paused for user interaction[p454] or paused for in-band content[p454].

A media element[p430] is said to be **potentially playing** when its paused[p453] attribute is false, the element has not ended playback[p453], playback has not stopped due to errors[p453], and the element is not a blocked media element[p453].

Note

A waiting[p485] DOM event can be fired[p451] as a result of an element that is potentially playing[p453] stopping playback due to its readyState[p452] attribute changing to a value lower than HAVE_FUTURE_DATA[p450].

A media element[p430] is said to be **eligible for autoplay** when all of the following are true:

- its can autoplay flag[p435] is true;
- its paused[p453] attribute is true;
- it has an autoplay[p452] attribute specified;
- if it is an audio[p425] or video[p420] element and its lazy loading attribute[p105] is either in the Eager[p105] state or has started loading while in the Lazy[p105] state, or scripting is disabled[p1152] for the element;
- its node document's active sandboxing flag set[p959] does not have the sandboxed automatic features browsing context flag[p958] set; and
- its node document is allowed to use[p412] the "autoplay[p79]" feature.

A media element[p430] 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[p430]'s Window[p965] object has transient activation[p868], but an exception could be made to allow playback while muted[p482].

A media element[p430] is said to have **ended playback** when:

- The element's readyState[p452] attribute is HAVE_METADATA[p450] or greater, and
- Either:
  - The current playback position[p448] is the end of the media resource[p432], and
  - The direction of playback[p457] is forwards, and
  - The media element[p430] does not have a loop[p450] attribute specified.

  Or:
  - The current playback position[p448] is the earliest possible position[p449], and
  - The direction of playback[p457] is backwards.

The **ended** attribute must return true if, the last time the event loop[p1193] reached step 1[p1196], the media element[p430] had ended playback[p453] and the direction of playback[p457] was forwards, and false otherwise.

A media element[p430] is said to have **stopped due to errors** when the element's readyState[p452] attribute is HAVE_METADATA[p450] or greater, and the user agent encounters a non-fatal error[p445] during the processing of the media data[p431], and due to that error, is not able to play the content at the current playback position[p448].

A media element[p430] is said to have **paused for user interaction** when its paused[p453] attribute is false, the readyState[p452] attribute is either HAVE_FUTURE_DATA[p450] or HAVE_ENOUGH_DATA[p450], and the user agent has reached a point in the media resource[p432] where the user has to make a selection for the resource to continue.

It is possible for a media element[p430] to have both ended playback[p453] and paused for user interaction[p454] at the same time.

When a media element[p430] that is potentially playing[p453] stops playing because it has paused for user interaction[p454], the user agent must queue a media element task[p432] given the media element[p430] to fire an event named timeupdate[p485] at the element.

A media element[p430] is said to have **paused for in-band content** when its paused[p453] attribute is false, the readyState[p452] attribute is either HAVE_FUTURE_DATA[p450] or HAVE_ENOUGH_DATA[p450], and the user agent has suspended playback of the media resource[p432] in order to play content that is temporally anchored to the media resource[p432] and has a nonzero length, or to play content that is temporally anchored to a segment of the media resource[p432] but has a length longer than that segment.

Example

One example of when a media element[p430] would be paused for in-band content[p454] is when the user agent is playing audio descriptions[p428] 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[p468] and the text track cue end time[p468].

When the current playback position[p448] reaches the end of the media resource[p432] when the direction of playback[p457] is forwards, then the user agent must follow these steps:

1. If the media element[p430] has a loop[p450] attribute specified, then seek[p460] to the earliest possible position[p449] of the media resource[p432] and return.
2. As defined above, the ended[p453] IDL attribute starts returning true once the event loop[p1193] returns to step 1[p1196].
3. Queue a media element task[p432] given the media element[p430] and the following steps:
    1. Fire an event named timeupdate[p485] at the media element[p430].
    2. If the media element[p430] has ended playback[p453], the direction of playback[p457] is forwards, and paused[p453] is false:
        1. Set the paused[p453] attribute to true.
        2. Fire an event named pause[p485] at the media element[p430].
        3. Take pending play promises[p455] and reject pending play promises[p455] with the result and an "AbortError" DOMException.
    3. Fire an event named ended[p485] at the media element[p430].

When the current playback position[p448] reaches the earliest possible position[p449] of the media resource[p432] when the direction of playback[p457] is backwards, then the user agent must only queue a media element task[p432] given the media element[p430] to fire an event named timeupdate[p485] at the element.

Note

The word "reaches" here does not imply that the current playback position[p448] needs to have changed during normal playback; it could be via seeking[p460], for instance.

The **defaultPlaybackRate** attribute gives the desired speed at which the media resource[p432] 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[p454] is used by the user agent when it exposes a user interface to the user[p481].

The **playbackRate** attribute gives the effective playback rate, which is the speed at which the media resource[p432] plays, as a multiple of its intrinsic speed. If it is not equal to the defaultPlaybackRate[p454], 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 "NotSupportedError" DOMException.
2. Set playbackRate[p454] to the new value, and if the element is potentially playing[p453], change the playback speed.

When the defaultPlaybackRate[p454] or playbackRate[p454] 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[p432] given the media element[p430] to fire an event named ratechange[p485] at the media element[p430]. 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** getter steps are to return a new normalized TimeRanges object[p484] that represents the ranges of points on the media timeline[p447] of this's media resource[p432] reached through the usual monotonic increase of this's current playback position[p448] during normal playback, if any.

⚠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[p430] has a **list of pending play promises**, which must initially be empty.

To **take pending play promises** for a media element[p430], the user agent must run the following steps:

1. Let *promises* be an empty list of promises.
2. Copy the media element[p430]'s list of pending play promises[p455] to *promises*.
3. Clear the media element[p430]'s list of pending play promises[p455].
4. Return *promises*.

To **resolve pending play promises** for a media element[p430] 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[p430] with a list of promises *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[p430], the user agent must run the following steps:

1. Take pending play promises[p455] and let *promises* be the result.
2. Queue a media element task[p432] given the element and the following steps:
    1. Fire an event named playing[p485] at the element.
    2. Resolve pending play promises[p455] with *promises*.

When the **play()** method on a media element[p430] is invoked, the user agent must run the following steps:

1. If the media element[p430] is not allowed to play[p453], then return a promise rejected with a "NotAllowedError" DOMException.
2. If the media element[p430]'s error[p432] attribute is not null and its code[p432] is MEDIA_ERR_SRC_NOT_SUPPORTED[p433], then return a promise rejected with a "NotSupportedError" DOMException.

    Note

    This means that the dedicated media source failure steps[p439] have run. Playback is not possible until the media element load algorithm[p435] clears the error[p432] attribute.

3. Let *resumptionSteps* be the media element[p430]'s lazy load resumption steps[p105].
4. If *resumptionSteps* is not null:

1. Set the media element[p430]'s lazy load resumption steps[p105] to null.
2. Invoke *resumptionSteps*.

5. Let *promise* be a new promise and append *promise* to the list of pending play promises[p455].
6. Run the internal play steps[p456] for the media element[p430].
7. Return *promise*.

The **internal play steps** for a media element[p430] are as follows:

1. If the media element[p430]'s networkState[p435] attribute has the value NETWORK_EMPTY[p435], invoke the media element[p430]'s resource selection algorithm[p436].
2. If the playback has ended[p453] and the direction of playback[p457] is forwards, seek[p460] to the earliest possible position[p449] of the media resource[p432].

   Note

   This will cause[p461] the user agent to queue a media element task[p432] given the media element[p430] to fire an event named timeupdate[p485] *at the media element[p430].*
3. If the media element[p430]'s paused[p453] attribute is true:
   1. Change the value of paused[p453] to false.
   2. If the show poster flag[p449] is true, set the element's show poster flag[p449] to false and run the *time marches on[p458]* steps.
   3. Queue a media element task[p432] given the media element[p430] to fire an event named play[p485] at the element.
   4. If the media element[p430]'s readyState[p452] attribute has the value HAVE_NOTHING[p450], HAVE_METADATA[p450], or HAVE_CURRENT_DATA[p450], queue a media element task[p432] given the media element[p430] to fire an event named waiting[p485] at the element.

      Otherwise, the media element[p430]'s readyState[p452] attribute has the value HAVE_FUTURE_DATA[p450] or HAVE_ENOUGH_DATA[p450]: notify about playing[p455] for the element.
4. Otherwise, if the media element[p430]'s readyState[p452] attribute has the value HAVE_FUTURE_DATA[p450] or HAVE_ENOUGH_DATA[p450], take pending play promises[p455] and queue a media element task[p432] given the media element[p430] to resolve pending play promises[p455] with the result.

   Note

   The media element is already playing. However, it's possible that promise *will be rejected[p455] before the queued task is run.*
5. Set the media element[p430]'s can autoplay flag[p435] to false.

When the **pause()** method is invoked, and when the user agent is required to pause the media element[p430], the user agent must run the following steps:

1. If the media element[p430]'s networkState[p435] attribute has the value NETWORK_EMPTY[p435], invoke the media element[p430]'s resource selection algorithm[p436].
2. Run the internal pause steps[p456] for the media element[p430].

The **internal pause steps** for a media element[p430] are as follows:

1. Set the media element[p430]'s can autoplay flag[p435] to false.
2. If the media element[p430]'s paused[p453] attribute is false, run the following steps:
   1. Change the value of paused[p453] to true.
   2. Take pending play promises[p455] and let *promises* be the result.

3. Queue a media element task[p432] given the media element[p430] and the following steps:

    1. Fire an event named timeupdate[p485] at the element.

    2. Fire an event named pause[p485] at the element.

    3. Reject pending play promises[p455] with *promises* and an "AbortError" DOMException.

4. Set the official playback position[p448] to the current playback position[p448].

If the element's playbackRate[p454] is positive or zero, then the **direction of playback** is forwards. Otherwise, it is backwards.

When a media element[p430] is potentially playing[p453] and its Document[p135] is a fully active[p1051] Document[p135], its current playback position[p448] must increase monotonically at the element's playbackRate[p454] units of media time per unit time of the media timeline[p447]'s clock. (This specification always refers to this as an *increase*, but that increase could actually be a *de*crease if the element's playbackRate[p454] is negative.)

Note

The element's playbackRate[p454] can be 0.0, in which case the current playback position[p448] doesn't move, despite playback not being paused (paused[p453] doesn't become true, and the pause[p485] event doesn't fire).

Note

This specification doesn't define how the user agent achieves the appropriate playback rate — depending on the protocol and media available, it is plausible that the user agent could negotiate with the server to have the server provide the media data at the appropriate rate, so that (except for the period between when the rate is changed and when the server updates the stream's playback rate) the client doesn't actually have to drop or interpolate any frames.

Any time the user agent provides a stable state[p1201], the official playback position[p448] must be set to the current playback position[p448].

If the element's playbackRate[p454] is not 1.0 and preservesPitch[p455] 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[p430] is potentially playing[p453], its audio data played must be synchronized with the current playback position[p448], at the element's effective media volume[p482]. The user agent must play the audio from audio tracks that were enabled when the event loop[p1193] last reached step 1[p1196].

When a media element[p430] is not potentially playing[p453], audio must not play for the element.

Media elements[p430] that are potentially playing[p453] 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 stalled waiting for content to buffer, or it could be still buffering, but with a suspend[p484] event listener that begins playback. Even a media element whose media resource[p432] has no audio tracks could eventually play audio again if it had an event listener that changes the media resource[p432].

Each media element[p430] has a **list of newly introduced cues**, which must be initially empty. Whenever a text track cue[p468] is added to the list of cues[p467] of a text track[p466] that is in the list of text tracks[p466] for a media element[p430], that cue[p468] must be added to the media element[p430]'s list of newly introduced cues[p457]. Whenever a text track[p466] is added to the list of text tracks[p466] for a media element[p430], all of the cues[p468] in that text track[p466]'s list of cues[p467] must be added to the media element[p430]'s list of newly introduced cues[p457]. When a media element[p430]'s list of newly introduced cues[p457] has new cues added while the media element[p430]'s show poster flag[p449] is not set, then the user agent must run the *time marches on*[p458] steps.

When a text track cue[p468] is removed from the list of cues[p467] of a text track[p466] that is in the list of text tracks[p466] for a media element[p430], and whenever a text track[p466] is removed from the list of text tracks[p466] of a media element[p430], if the media element[p430]'s show poster flag[p449] is not set, then the user agent must run the *time marches on*[p458] steps.

When the current playback position[p448] of a media element[p430] changes (e.g. due to playback or seeking), the user agent must run the

time marches on[p458] 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[p448] 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[p468] to be skipped over as the user agent rushes ahead to "catch up", so these cues will not appear in the `activeCues`[p476] *list.*

The ***time marches on*** steps are as follows:

1. Let *current cues* be a list of cues[p468], initialized to contain all the cues[p468] of all the hidden[p467] or showing[p467] text tracks[p466] of the media element[p430] (not the disabled[p467] ones) whose start times[p468] are less than or equal to the current playback position[p448] and whose end times[p468] are greater than the current playback position[p448].

2. Let *other cues* be a list of cues[p468], initialized to contain all the cues[p468] of hidden[p467] and showing[p467] text tracks[p466] of the media element[p430] that are not present in *current cues*.

3. Let *last time* be the current playback position[p448] at the time this algorithm was last run for this media element[p430], if this is not the first time it has run.

4. If the current playback position[p448] 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[p468] in *other cues* whose start times[p468] are greater than or equal to *last time* and whose end times[p468] are less than or equal to the current playback position[p448]. Otherwise, let *missed cues* be an empty list.

5. Remove all the cues[p468] in *missed cues* that are also in the media element[p430]'s list of newly introduced cues[p457], and then empty the element's list of newly introduced cues[p457].

6. If the time was reached through the usual monotonic increase of the current playback position[p448] during normal playback, and if the user agent has not fired a `timeupdate`[p485] 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[p432] given the media element[p430] to fire an event named `timeupdate`[p485] 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[p448].)

   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[p468] in *current cues* have their text track cue active flag[p469] set, none of the cues[p468] in *other cues* have their text track cue active flag[p469] set, and *missed cues* is empty, then return.

8. If the time was reached through the usual monotonic increase of the current playback position[p448] during normal playback, and there are cues[p468] in *other cues* that have their text track cue pause-on-exit flag[p468] set and that either have their text track cue active flag[p469] set or are also in *missed cues*, then immediately[p44] pause[p456] the media element[p430].

   Note

   In the other cases, such as explicit seeks, playback is not paused by going past the end time of a cue[p468], even if that cue[p468] has its text track cue pause-on-exit flag[p468] set.

9. Let *events* be a list of tasks[p1194], initially empty. Each task[p1194] in this list will be associated with a text track[p466], a text track cue[p468], and a time, which are used to sort the list before the tasks[p1194] are queued.

   Let *affected tracks* be a list of text tracks[p466], initially empty.

   When the steps below say to **prepare an event** named *event* for a text track cue[p468] *target* with a time *time*, the user agent must run these steps:

   1. Let *track* be the text track[p466] with which the text track cue[p468] *target* is associated.

   2. Create a task[p1194] to fire an event named *event* at *target*.

   3. Add the newly created task[p1194] to *events*, associated with the time *time*, the text track[p466] *track*, and the text track

cue[p468] *target*.

4. Add *track* to *affected tracks*.

10. For each text track cue[p468] in *missed cues*, prepare an event[p458] named enter[p486] for the TextTrackCue[p478] object with the text track cue start time[p468].

11. For each text track cue[p468] in *other cues* that either has its text track cue active flag[p469] set or is in *missed cues*, prepare an event[p458] named exit[p486] for the TextTrackCue[p478] object with the later of the text track cue end time[p468] and the text track cue start time[p468].

12. For each text track cue[p468] in *current cues* that does not have its text track cue active flag[p469] set, prepare an event[p458] named enter[p486] for the TextTrackCue[p478] object with the text track cue start time[p468].

13. Sort the tasks[p1194] in *events* in ascending time order (tasks[p1194] with earlier times first).

    Further sort tasks[p1194] in *events* that have the same time by the relative text track cue order[p469] of the text track cues[p468] associated with these tasks[p1194].

    Finally, sort tasks[p1194] in *events* that have the same time and same text track cue order[p469] by placing tasks[p1194] that fire enter[p486] events before those that fire exit[p486] events.

14. Queue a media element task[p432] given the media element[p430] for each task[p1194] in *events*, in list order.

15. Sort *affected tracks* in the same order as the text tracks[p466] appear in the media element[p430]'s list of text tracks[p466], and remove duplicates.

16. For each text track[p466] in *affected tracks*, in the list order, queue a media element task[p432] given the media element[p430] to fire an event named cuechange[p486] at the TextTrack[p474] object, and, if the text track[p466] has a corresponding track[p427] element, to then fire an event named cuechange[p486] at the track[p427] element as well.

17. Set the text track cue active flag[p469] of all the cues[p468] in the *current cues*, and unset the text track cue active flag[p469] of all the cues[p468] in the *other cues*.

18. Run the rules for updating the text track rendering[p467] of each of the text tracks[p466] in *affected tracks* that are showing[p467], providing the text track[p466]'s text track language[p467] as the fallback language if it is not the empty string. For example, for text tracks[p466] based on WebVTT, the rules for updating the display of WebVTT text tracks. [WEBVTT][p1588]

For the purposes of the algorithm above, a text track cue[p468] is considered to be part of a text track[p466] only if it is listed in the text track list of cues[p467], not merely if it is associated with the text track[p466].

Note

If the media element[p430]'s node document stops being a fully active[p1051] document, then the playback will stop[p457] until the document is active again.

When a media element[p430] is removed from a Document[p47], the user agent must run the following steps:

1. Await a stable state[p1201], allowing the task[p1194] that removed the media element[p430] from the Document[p135] to continue. The synchronous section[p1201] consists of all the remaining steps of this algorithm. (Steps in the synchronous section[p1201] are marked with ⌛.)

2. ⌛ If the media element[p430] is in a document, return.

3. ⌛ Run the internal pause steps[p456] for the media element[p430].

4.8.11.9 Seeking §[p459]

For web developers (non-normative)

***media*.seeking**[p460]

Returns true if the user agent is currently seeking.

***media*.seekable**[p461]

Returns a TimeRanges[p483] object that represents the ranges of the media resource[p432] 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(*time*)** method must seek to the time given by *time*, with the *approximate-for-speed* flag set.

When the user agent is required to **seek** to a particular *new playback position* in the media resource, optionally with the *approximate-for-speed* flag set, it means that the user agent must run the following steps. This algorithm interacts closely with the event loop mechanism; in particular, it has a synchronous section (which is triggered as part of the event loop algorithm). Steps in that section are marked with ⌛.

1. Set the media element's show poster flag to false.
2. If the media element's readyState is HAVE_NOTHING, 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[p449] attribute returns the official playback position[p448], not the current playback position[p448], 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[p431] 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[p1201]. The synchronous section[p1201] consists of all the remaining steps of this algorithm. (Steps in the synchronous section[p1201] are marked with ⌛.)

14. ⌛ Set the seeking[p460] IDL attribute to false.

15. ⌛ Run the time marches on[p458] steps.

16. ⌛ Queue a media element task[p432] given the media element[p430] to fire an event named timeupdate[p485] at the element.

17. ⌛ Queue a media element task[p432] given the media element[p430] to fire an event named seeked[p485] at the element.

The **seekable** getter steps are to return a new normalized TimeRanges object[p484] that represents the ranges of this's media resource[p432], if any, that the user agent is able to seek to.

Note

If the user agent can seek to anywhere in the media resource[p432], 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[p449], typically zero), and whose end is the same as the time of the first frame plus the duration[p449] attribute's value (which would equal the time of the last frame, and might be positive Infinity).

Note

The range might be continuously changing, e.g. if the user agent is buffering a sliding window on an infinite stream. This is the behavior seen with DVRs viewing live TV, for instance.

⚠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[p432] might be internally scripted or interactive. Thus, a media element[p430] could play in a non-linear fashion. If this happens, the user agent must act as if the algorithm for seeking[p460] was used whenever the current playback position[p448] changes in a discontinuous fashion (so that the relevant events fire).

4.8.11.10 Media resources with multiple media tracks §[p461]

A media resource[p432] can have multiple embedded audio and video tracks. For example, in addition to the primary video and audio tracks, a media resource[p432] could have foreign-language dubbed dialogues, director's commentaries, audio descriptions, alternative angles, or sign-language overlays.

For web developers (non-normative)

***media*.audioTracks** [p462]

Returns an AudioTrackList [p462] object representing the audio tracks available in the media resource [p432].

***media*.videoTracks** [p462]

Returns a VideoTrackList [p462] object representing the video tracks available in the media resource [p432].

The **audioTracks** attribute of a media element [p430] must return a live [p48] AudioTrackList [p462] object representing the audio tracks available in the media element [p430]'s media resource [p432].

The **videoTracks** attribute of a media element [p430] must return a live [p48] VideoTrackList [p462] object representing the video tracks available in the media element [p430]'s media resource [p432].

Note

There are only ever one AudioTrackList [p462] object and one VideoTrackList [p462] object per media element [p430], even if another media resource [p432] is loaded into the element: the objects are reused. (The AudioTrack [p462] and VideoTrack [p462] objects are not, though.)

4.8.11.10.1 AudioTrackList [p462] *and* VideoTrackList [p462] *objects* § [p462]

The AudioTrackList [p462] and VideoTrackList [p462] 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)

***media*.audioTracks[p462].length[p464]**

***media*.videoTracks[p462].length[p464]**

Returns the number of tracks in the list.

***audioTrack* = *media*.audioTracks[p462][*index*]**

***videoTrack* = *media*.videoTracks[p462][*index*]**

Returns the specified AudioTrack[p462] or VideoTrack[p462] object.

***audioTrack* = *media*.audioTracks[p462].getTrackById[p464](*id*)**

***videoTrack* = *media*.videoTracks[p462].getTrackById[p464](*id*)**

Returns the AudioTrack[p462] or VideoTrack[p462] object with the given identifier, or null if no track has that identifier.

***audioTrack*.id[p464]**

***videoTrack*.id[p464]**

Returns the ID of the given track. 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()` method.

***audioTrack*.kind[p464]**

***videoTrack*.kind[p464]**

Returns the category the given track falls into. The possible track categories[p464] are given below.

***audioTrack*.label[p465]**

***videoTrack*.label[p465]**

Returns the label of the given track, if known, or the empty string otherwise.

***audioTrack*.language[p465]**

***videoTrack*.language[p465]**

Returns the language of the given track, if known, or the empty string otherwise.

***audioTrack*.enabled[p465] [= *value*]**

Returns true if the given track is active, and false otherwise.

Can be set, to change whether the track is enabled or not. If multiple audio tracks are enabled simultaneously, they are mixed.

***media*.videoTracks[p462].selectedIndex[p465]**

Returns the index of the currently selected track, if any, or −1 otherwise.

***videoTrack*.selected[p465] [= *value*]**

Returns true if the given track is active, and false otherwise.

Can be set, to change whether the track is selected or not. Either zero or one video track is selected; selecting a new track while a previous one is selected will unselect the previous one.

An AudioTrackList[p462] 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[p462] object.

A VideoTrackList[p462] 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[p462] object.

Tracks in AudioTrackList[p462] and VideoTrackList[p462] objects must be consistently ordered. If the media resource[p432] 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[p432]. The order used is called the *natural order* of the list.

Note

Each track in one of these objects thus has an index; the first has the index 0, and each subsequent track is numbered one higher than the previous one. If a media resource[p432] dynamically adds or removes audio or video tracks, then the indices of the tracks will change dynamically. If the media resource[p432] changes entirely, then all the previous tracks will be removed and replaced with new tracks.

The AudioTrackList[p462] **length** and VideoTrackList[p462] **length** attribute getters must return the number of tracks represented by their objects at the time of getting.

The supported property indices of AudioTrackList[p462] and VideoTrackList[p462] 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[p462] or VideoTrackList[p462] 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[p462] or VideoTrackList[p462] object *list*, the user agent must return the AudioTrack[p462] or VideoTrack[p462] object that represents the *index*th track in *list*.

The AudioTrackList[p462] **getTrackById(*id*)** and VideoTrackList[p462] **getTrackById(*id*)** methods must return the first AudioTrack[p462] or VideoTrack[p462] object (respectively) in the AudioTrackList[p462] or VideoTrackList[p462] object (respectively) whose identifier is equal to the value of the *id* argument (in the natural order of the list, as defined above). When no tracks match the given argument, the methods must return null.

The AudioTrack[p462] and VideoTrack[p462] objects represent specific tracks of a media resource[p432]. 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[p432]'s AudioTrackList[p462] or VideoTrackList[p462] objects, those aspects do not change.

In addition, AudioTrack[p462] objects can each be enabled or disabled; this is the audio track's *enabled state*. When an AudioTrack[p462] is created, its *enabled state* must be set to false (disabled). The resource fetch algorithm[p440] can override this.

Similarly, a single VideoTrack[p462] object per VideoTrackList[p462] object can be selected, this is the video track's *selection state*. When a VideoTrack[p462] is created, its *selection state* must be set to false (not selected). The resource fetch algorithm[p440] can override this.

The AudioTrack[p462] **id** and VideoTrack[p462] **id** attributes must return the identifier of the track, if it has one, or the empty string otherwise. If the media resource[p432] 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. [INBAND][p1583]

Example

For example, in Ogg files, this would be the Name header field of the track. [OGGSKELETONHEADERS][p1584]

The AudioTrack[p462] **kind** and VideoTrack[p462] **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[p432]. 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[p462] objects if they are appropriate for audio, and must only be returned for VideoTrack[p462] 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][p1584] [DASH][p1582] [WEBMCG][p1588] [INBAND][p1583]

Return values for AudioTrack[p462]'s kind[p464] and VideoTrack[p462]'s kind[p464]

Category	Definition	Applies to...	Examples
"**alternative**"	A possible alternative to the main track, e.g. a different take of a song (audio), or a different angle (video).	Audio and video.	Ogg: "audio/alternate" or "video/alternate"; DASH: "alternate" without "main" and "commentary" roles, and, for audio, without the "dub" role (other roles ignored).
"**captions**"	A version of the main video track with captions burnt in. (For legacy content; new content would use text tracks.)	Video only.	DASH: "caption" and "main" roles together (other roles ignored).
"**descriptions**"	An audio description of a video track.	Audio only.	Ogg: "audio/audiodesc".
"**main**"	The primary audio or video track.	Audio and video.	Ogg: "audio/main" or "video/main"; WebM: the "FlagDefault" element is set; DASH: "main" role without "caption", "subtitle", and "dub" roles (other roles ignored).
"**main-desc**"	The primary audio track, mixed with audio descriptions.	Audio only.	AC3 audio in MPEG-2 TS: bsmod=2 and full_svc=1.
"**sign**"	A sign-language interpretation of an audio track.	Video	Ogg: "video/sign".

Category	Definition	Applies to...	Examples
		only.	
"**subtitles**"	A version of the main video track with subtitles burnt in. (For legacy content; new content would use text tracks.)	Video only.	DASH: "subtitle" and "main" roles together (other roles ignored).
"**translation**"	A translated version of the main audio track.	Audio only.	Ogg: "audio/dub". DASH: "dub" and "main" roles together (other roles ignored).
"**commentary**"	Commentary on the primary audio or video track, e.g. a director's commentary.	Audio and video.	DASH: "commentary" role without "main" role (other roles ignored).
"" (empty string)	No explicit kind, or the kind given by the track's metadata is not recognized by the user agent.	Audio and video.	

The AudioTrack[p462] **label** and VideoTrack[p462] **label** attributes must return the label of the track, if it has one, or the empty string otherwise. [INBAND][p1583]

The AudioTrack[p462] **language** and VideoTrack[p462] **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[p432]'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][p1583]

The AudioTrack[p462] **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[p462] object, then the track being enabled or disabled has no effect beyond changing the value of the attribute on the AudioTrack[p462] object.)

Whenever an audio track in an AudioTrackList[p462] that was disabled is enabled, and whenever one that was enabled is disabled, the user agent must queue a media element task[p432] given the media element[p430] to fire an event named change[p486] at the AudioTrackList[p462] object.

An audio track that has no data for a particular position on the media timeline[p447], or that does not exist at that position, must be interpreted as being silent at that point on the timeline.

The VideoTrackList[p462] **selectedIndex** attribute must return the index of the currently selected track, if any. If the VideoTrackList[p462] object does not currently represent any tracks, or if none of the tracks are selected, it must instead return −1.

The VideoTrack[p462] **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[p462], then all the other VideoTrack[p462] objects in that list must be unselected. (If the track is no longer in a VideoTrackList[p462] object, then the track being selected or unselected has no effect beyond changing the value of the attribute on the VideoTrack[p462] object.)

Whenever a track in a VideoTrackList[p462] that was previously not selected is selected, and whenever the selected track in a VideoTrackList[p462] is unselected without a new track being selected in its stead, the user agent must queue a media element task[p432] given the media element[p430] to fire an event named change[p486] at the VideoTrackList[p462] object. This task[p1194] must be queued[p1195] before the task[p1194] that fires the resize[p485] event, if any.

A video track that has no data for a particular position on the media timeline[p447] 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[p432] but was simply transparent until one hour in.

The following are the event handlers[p1206] (and their corresponding event handler event types[p1209]) that must be supported, as event handler IDL attributes[p1207], by all objects implementing the AudioTrackList[p462] and VideoTrackList[p462] interfaces:

Event handler[p1206]	Event handler event type[p1209]
onchange	change[p486]

Event handler[p1206]	Event handler event type[p1209]
onaddtrack	addtrack[p486]
onremovetrack	removetrack[p486]

4.8.11.10.2 Selecting specific audio and video tracks declaratively §[p46 6]

The audioTracks[p462] and videoTracks[p462] 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[p432]. The format of the fragment depends on the MIME type of the media resource[p432]. [RFC2046][p1585] [URL][p1587]

Example

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.

```
<video src="myvideo#track=Alternative"></video>
```

4.8.11.11 Timed text tracks §[p46 6]

4.8.11.11.1 Text track model §[p46 6]

A media element[p430] can have a group of associated **text tracks**, known as the media element[p430]'s **list of text tracks**. The text tracks[p466] are sorted as follows:

1. The text tracks[p466] corresponding to track[p427] element children of the media element[p430], in tree order.
2. Any text tracks[p466] added using the addTextTrack()[p475] method, in the order they were added, oldest first.
3. Any media-resource-specific text tracks[p469] (text tracks[p466] corresponding to data in the media resource[p432]), in the order defined by the media resource[p432]'s format specification.

A text track[p466] is a TextTrack[p474] object and 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[p466] can change dynamically, in the case of a text track[p466] corresponding to a track[p427] element.

A label

This is a human-readable string intended to identify the track for the user.

The label of a track[p466] can change dynamically, in the case of a text track[p466] corresponding to a track[p427] element.

When a text track label[p466] 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[p432] 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[p466] 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[p469].

A language

This is a string (a BCP 47 language tag) representing the language of the text track's cues. [BCP47][p1579]

The language of a text track[p467] can change dynamically, in the case of a text track[p466] corresponding to a `track`[p427] element.

An identifier

A string.

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[p101], network error, unknown text track format). Some or all of the cues are likely missing and will not be obtained.

The readiness state[p467] of a text track[p466] 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[p466] is `subtitles`[p466] or `captions`[p466], the cues are being overlaid on the video as appropriate; for text tracks whose kind[p466] is `descriptions`[p466], the user agent is making the cues available to the user in a non-visual fashion; and for text tracks whose kind[p466] is `chapters`[p466], the user agent is making available to the user a mechanism by which the user can navigate to any point in the media resource[p432] by selecting a cue.

A list of zero or more cues

A list of text track cues[p468], along with **rules for updating the text track rendering**. For example, for WebVTT, the rules for updating the display of WebVTT text tracks. [WEBVTT][p1588]

The list of cues of a text track[p467] can change dynamically, either because the text track[p466] has not yet been loaded[p467] or is still loading[p467], or due to DOM manipulation.

Each media element[p430] 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[p430], the user agent must add to the element's list of pending text tracks[p468] each text track[p466] in the element's list of text tracks[p466] whose text track mode[p467] is not disabled[p467] and whose text track readiness state[p467] is loading[p467].

Whenever a `track`[p427] element's parent node changes, the user agent must remove the corresponding text track[p466] from any list of pending text tracks[p468] that it is in.

Whenever a text track[p466]'s text track readiness state[p467] changes to either loaded[p467] or failed to load[p467], the user agent must remove it from any list of pending text tracks[p468] that it is in.

When a media element[p430] is created by an HTML parser[p1368] or XML parser[p1484], the user agent must set the element's blocked-on-parser[p468] flag to true. When a media element[p430] is popped off the stack of open elements[p1383] of an HTML parser[p1368] or XML parser[p1484], the user agent must honor user preferences for automatic text track selection[p471], populate the list of pending text tracks[p468], and set the element's blocked-on-parser[p468] flag to false.

The text tracks[p466] of a media element[p430] are **ready** when both the element's list of pending text tracks[p468] is empty and the element's blocked-on-parser[p468] flag is false.

Each media element[p430] has a **pending text track change notification flag**, which must initially be unset.

Whenever a text track[p466] that is in a media element[p430]'s list of text tracks[p466] has its text track mode[p467] change value, the user agent must run the following steps for the media element[p430]:

1. If the media element[p430]'s pending text track change notification flag[p468] is set, return.
2. Set the media element[p430]'s pending text track change notification flag[p468].
3. Queue a media element task[p432] given the media element[p430] to run these steps:
    1. Unset the media element[p430]'s pending text track change notification flag[p468].
    2. Fire an event named `change`[p486] at the media element[p430]'s `textTracks`[p474] attribute's `TextTrackList`[p474] object.
4. If the media element[p430]'s show poster flag[p449] is not set, run the *time marches on*[p458] steps.

The task source[p1194] for the tasks[p1194] listed in this section is the DOM manipulation task source[p1203].

A **text track cue** is the unit of time-sensitive data in a text track[p466], 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[p468] 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[p431] 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[p431] to which the cue applies, or positive Infinity for an unbounded text track cue[p469].

A pause-on-exit flag

A boolean indicating whether playback of the media resource[p432] 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][p1588]

An **unbounded text track cue** is a text track cue with a text track cue end time[p468] set to positive Infinity. An active unbounded text track cue[p469] cannot become inactive through the usual monotonic increase of the current playback position[p448] during normal playback (e.g. a metadata cue for a chapter in a live event with no announced end time.)

Note

The text track cue start time[p468] and text track cue end time[p468] can be negative. (The current playback position[p448] can never be negative, though, so cues entirely before time zero cannot be active.)

Each text track cue[p468] has a corresponding TextTrackCue[p478] object (or more specifically, an object that inherits from TextTrackCue[p478] — for example, WebVTT cues use the VTTCue interface). A text track cue[p468]'s in-memory representation can be dynamically changed through this TextTrackCue[p478] API. [WEBVTT][p1588]

A text track cue[p468] is associated with rules for updating the text track rendering[p467], as defined by the specification for the specific kind of text track cue[p468]. These rules are used specifically when the object representing the cue is added to a TextTrack[p474] object using the addCue()[p476] method.

In addition, each text track cue[p468] 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[p468] is removed from its text track[p466]'s text track list of cues[p467]; whenever the text track[p466] itself is removed from its media element[p430]'s list of text tracks[p466] or has its text track mode[p467] changed to disabled[p467]; and whenever the media element[p430]'s readyState[p452] is changed back to HAVE_NOTHING[p450]. When the flag is unset in this way for one or more cues in text tracks[p466] that were showing[p467] 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[p467] of those text tracks[p466]. For example, for text tracks[p466] based on WebVTT, the rules for updating the display of WebVTT text tracks. [WEBVTT][p1588]

The *display state*

This is used as part of the rendering model, to keep cues in a consistent position. It must initially be empty. Whenever the text track cue active flag[p469] is unset, the user agent must empty the text track cue display state[p469].

The text track cues[p468] of a media element[p430]'s text tracks[p466] are ordered relative to each other in the **text track cue order**, which is determined as follows: first group the cues[p468] by their text track[p466], with the groups being sorted in the same order as their text tracks[p466] appear in the media element[p430]'s list of text tracks[p466]; then, within each group, cues[p468] must be sorted by their start time[p468], earliest first; then, any cues[p468] with the same start time[p468] must be sorted by their end time[p468], latest first; and finally, any cues[p468] with identical end times[p468] must be sorted in the order they were last added to their respective text track list of cues[p467], oldest first (so e.g. for cues from a WebVTT file, that would initially be the order in which the cues were listed in the file). [WEBVTT][p1588]

4.8.11.11.2 Sourcing in-band text tracks §[p469]

A **media-resource-specific text track** is a text track[p466] that corresponds to data found in the media resource[p432].

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[p432] is a video. Details for some legacy formats can be found in *Sourcing In-band Media Resource Tracks from Media Containers into HTML*. [INBAND][p1583]

When a media resource[p432] contains data that the user agent recognizes and supports as being equivalent to a text track[p466], the user agent runs[p446] 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[p466]. The text track[p466] is a media-resource-specific text track[p469].
2. Set the new text track[p466]'s kind[p466], label[p466], language[p467], and identifier[p467] 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[p466] must be set to the empty string.

   If the media resource[p432] is in a format that supports media fragment syntax, then the identifier[p467] 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.
3. Associate the text track list of cues[p467] with the rules for updating the text track rendering[p467] 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. [OGGSKELETONHEADERS]

   ↪ **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* bytes, expressed in hexadecimal using ASCII upper hex digits. [MPEG2]

   ↪ **If the media resource is an MPEG-4 file**

   Let *stsd box* be 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, or null if no such `stsd` box exists. If *stsd box* is null, or if *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 *stsd box* has a `mett` box, then the text track in-band metadata track dispatch type must be set to either 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 *stsd box*, or the empty string if that field is absent in that box. Otherwise, if *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 either 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 *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.

   Note

   For instance, if there are no other active subtitles, and this is a forced subtitle track (a subtitle track giving subtitles in the audio track's primary language, but only for audio that is actually in another language), then those subtitles might be activated here.

8. Add the new text track to the media element's list of text tracks.

9. Fire an event named `addtrack` at the media element's `textTracks` attribute's `TextTrackList` object, using `TrackEvent`, with the `track` attribute initialized to the text track.

4.8.11.11.3 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).

The text track kind is determined from the state of the element's `kind` attribute according to the following table; for a state given in a cell of the first column, the kind is the string given in the second column:

State	String
Subtitles	`subtitles`
Captions	`captions`
Descriptions	`descriptions`
Chapters metadata	`chapters`

State	String
Metadata[p428]	`metadata`[p466]

The text track label[p466] is the element's track label[p429].

The text track language[p467] is the element's track language[p429], if any; otherwise the empty string.

The text track identifier[p467] is the element's `id`[p162] attribute value, if any; otherwise the empty string.

As the `kind`[p428], `label`[p429], `srclang`[p428], and `id`[p162] attributes are set, changed, or removed, the text track[p466] must update accordingly, as per the definitions above.

Note

Changes to the track URL[p428] are handled in the algorithm below.

The text track readiness state[p467] is initially not loaded[p467], and the text track mode[p467] is initially disabled[p467].

The text track list of cues[p467] is initially empty. It is dynamically modified when the referenced file is parsed. Associated with the list are the rules for updating the text track rendering[p467] appropriate for the format in question; for WebVTT, this is the rules for updating the display of WebVTT text tracks. [WEBVTT][p1588]

When a `track`[p427] element's parent element changes and the new parent is a media element[p430], then the user agent must add the `track`[p427] element's corresponding text track[p466] to the media element[p430]'s list of text tracks[p466], and then queue a media element task[p432] given the media element[p430] to fire an event named `addtrack`[p486] at the media element[p430]'s `textTracks`[p474] attribute's `TextTrackList`[p474] object, using `TrackEvent`[p484], with the `track`[p484] attribute initialized to the text track[p466].

When a `track`[p427] element's parent element changes and the old parent was a media element[p430], then the user agent must remove the `track`[p427] element's corresponding text track[p466] from the media element[p430]'s list of text tracks[p466], and then queue a media element task[p432] given the media element[p430] to fire an event named `removetrack`[p486] at the media element[p430]'s `textTracks`[p474] attribute's `TextTrackList`[p474] object, using `TrackEvent`[p484], with the `track`[p484] attribute initialized to the text track[p466].

When a text track[p466] corresponding to a `track`[p427] element is added to a media element[p430]'s list of text tracks[p466], the user agent must queue a media element task[p432] given the media element[p430] to run the following steps for the media element[p430]:

1. If the element's blocked-on-parser[p468] flag is true, then return.
2. If the element's did-perform-automatic-track-selection[p468] flag is true, then return.
3. Honor user preferences for automatic text track selection[p471] for this element.

When the user agent is required to **honor user preferences for automatic text track selection** for a media element[p430], the user agent must run the following steps:

1. Perform automatic text track selection[p471] for `subtitles`[p466] and `captions`[p466].
2. Perform automatic text track selection[p471] for `descriptions`[p466].
3. If there are any text tracks[p466] in the media element[p430]'s list of text tracks[p466] whose text track kind[p466] is `chapters`[p466] or `metadata`[p466] that correspond to `track`[p427] elements with a `default`[p429] attribute set whose text track mode[p467] is set to disabled[p467], then set the text track mode[p467] of all such tracks to hidden[p467].
4. Set the element's did-perform-automatic-track-selection[p468] flag to true.

When the steps above say to **perform automatic text track selection** for one or more text track kinds[p466], it means to run the following steps:

1. Let *candidates* be a list consisting of the text tracks[p466] in the media element[p430]'s list of text tracks[p466] whose text track kind[p466] is one of the kinds that were passed to the algorithm, if any, in the order given in the list of text tracks[p466].
2. If *candidates* is empty, then return.
3. If any of the text tracks[p466] in *candidates* have a text track mode[p467] set to showing[p467], return.
4. If the user has expressed an interest in having a track from *candidates* enabled based on its text track kind[p466], text track

language[p467], and text track label[p466], then set its text track mode[p467] to showing[p467].

Note

For example, the user could have set a browser preference to the effect of "I want French captions whenever possible", or "If there is a subtitle track with 'Commentary' in the title, enable it", or "If there are audio description tracks available, enable one, ideally in Swiss German, but failing that in Standard Swiss German or Standard German".

Otherwise, if there are any text tracks[p466] in *candidates* that correspond to `track`[p427] elements with a `default`[p429] attribute set whose text track mode[p467] is set to disabled[p467], then set the text track mode[p467] of the first such track to showing[p467].

When a text track[p466] corresponding to a `track`[p427] element experiences any of the following circumstances, the user agent must start the `track` processing model[p472] for that text track[p466] and its `track`[p427] element:

- The `track`[p427] element is created.
- The text track[p466] has its text track mode[p467] changed.
- The `track`[p427] element's parent element changes and the new parent is a media element[p430].

When a user agent is to **start the `track` processing model** for a text track[p466] and its `track`[p427] element, it must run the following algorithm. This algorithm interacts closely with the event loop[p1193] mechanism; in particular, it has a synchronous section[p1201] (which is triggered as part of the event loop[p1193] algorithm). The steps in that section are marked with ⌛.

1. If another occurrence of this algorithm is already running for this text track[p466] and its `track`[p427] element, return, letting that other algorithm take care of this element.
2. If the text track[p466]'s text track mode[p467] is not set to one of hidden[p467] or showing[p467], then return.
3. If the text track[p466]'s `track`[p427] element does not have a media element[p430] as a parent, return.
4. Run the remainder of these steps in parallel[p44], allowing whatever caused these steps to run to continue.
5. *Top*: Await a stable state[p1201]. The synchronous section[p1201] consists of the following steps. (The steps in the synchronous section[p1201] are marked with ⌛.)
6. ⌛ Set the text track readiness state[p467] to loading[p467].
7. ⌛ Let *URL* be the track URL[p428] of the `track`[p427] element.
8. ⌛ If the `track`[p427] element's parent is a media element[p430], then let *corsAttributeState* be the state of the parent media element[p430]'s `crossorigin`[p433] content attribute. Otherwise, let *corsAttributeState* be No CORS[p103].
9. End the synchronous section[p1201], continuing the remaining steps in parallel[p44].
10. If *URL* is not the empty string:
    1. Let *request* be the result of creating a potential-CORS request[p102] given *URL*, "`track`", and *corsAttributeState*, and with the *same-origin fallback flag* set.
    2. Set *request*'s client to the `track`[p427] element's node document's relevant settings object[p1151].
    3. Set *request*'s initiator type to "`track`".
    4. Fetch *request*.

    The tasks[p1194] queued[p1194] by the fetching algorithm on the networking task source[p1204] 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[p467] being used for that parser's output. [WEBVTT][p1588]

Note

The appropriate parser will incrementally update the text track list of cues[p467] during these networking task source[p1204] tasks[p1194], 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. Implementers 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[p1195] on the DOM manipulation task source[p1203] given the media element[p430] to first change the text track readiness state[p467] to failed to load[p467] and then fire an event named `error`[p486] at the `track`[p427] 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[p1194] that is queued[p1195] on the networking task source[p1204] in which the aforementioned problem is found must change the text track readiness state[p467] to failed to load[p467] and fire an event named `error`[p486] at the `track`[p427] element.

If fetching does not fail, and the file was successfully processed, then the final task[p1194] that is queued[p1195] by the networking task source[p1204], after it has finished parsing the data, must change the text track readiness state[p467] to loaded[p467], and fire an event named `load`[p486] at the `track`[p427] element.

If, while fetching is ongoing, either:

- the track URL[p428] changes so that it is no longer equal to *URL*, while the text track mode[p467] is set to hidden[p467] or showing[p467]; or
- the text track mode[p467] changes to hidden[p467] or showing[p467], while the track URL[p428] is not equal to *URL*,

...then the user agent must abort fetching, discarding any pending tasks[p1194] generated by that algorithm (and in particular, not adding any cues to the text track list of cues[p467] after the moment the URL changed), and then queue an element task[p1195] on the DOM manipulation task source[p1203] given the `track`[p427] element that first changes the text track readiness state[p467] to failed to load[p467] and then fires an event named `error`[p486] at the `track`[p427] element.

11. Wait until the text track readiness state[p467] is no longer set to loading[p467].
12. Wait until the track URL[p428] is no longer equal to *URL*, at the same time as the text track mode[p467] is set to hidden[p467] or showing[p467].
13. Jump to the step labeled *top*.

Whenever a `track`[p427] element has its `src`[p428] attribute set, changed, or removed, the user agent must immediately[p44] empty the element's text track[p466]'s text track list of cues[p467]. (This also causes the algorithm above to stop adding cues from the resource being obtained using the previously given URL, if any.)

4.8.11.11.4 Guidelines for exposing cues in various formats as text track cues[p468] §[p473]

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[p466] model of HTML, each unit of timed data is converted to a text track cue[p468]. Where the mapping of the format's features to the aspects of a text track cue[p468] 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[p468] as defined above, as well as with the following constraints:

The text track cue identifier[p468]

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[p468]

Should be set to false.

4.8.11.11.5 Text track API §[p474]

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[p474].length**

Returns the number of text tracks[p466] associated with the media element[p430] (e.g. from track[p427] elements). This is the number of text tracks[p466] in the media element[p430]'s list of text tracks[p466].

***media*.textTracks[p474][*n*]**

Returns the *n*th text track[p466] in the media element[p430]'s list of text tracks[p466].

***textTrack* = *media*.textTracks[p474].getTrackById[p474](*id*)**

Returns the TextTrack[p474] object with the given identifier, or null if no track has that identifier.

A TextTrackList[p474] object represents a dynamically updating list of text tracks[p466] in a given order.

The **textTracks** attribute of media elements[p430] must return a TextTrackList[p474] object representing the text tracks[p466] in the media element[p430]'s list of text tracks[p466], in the same order as in the list of text tracks[p466].

The **length** attribute of a TextTrackList[p474] object must return the number of text tracks[p466] in the list represented by the TextTrackList[p474] object.

The supported property indices of a TextTrackList[p474] object at any instant are the numbers from zero to the number of text tracks[p466] in the list represented by the TextTrackList[p474] object minus one, if any. If there are no text tracks[p466] in the list, there are no supported property indices.

To determine the value of an indexed property of a TextTrackList[p474] object for a given index *index*, the user agent must return the *index*th text track[p466] in the list represented by the TextTrackList[p474] object.

The **getTrackById(*id*)** method must return the first TextTrack[p474] in the TextTrackList[p474] object whose id[p476] IDL attribute would return a value equal to the value of the *id* argument. When no tracks match the given argument, the method must return null.

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)

***textTrack* = *media*.addTextTrack[p475](*kind* [, *label* [, *language*]])**

Creates and returns a new TextTrack[p474] object, which is also added to the media element[p430]'s list of text tracks[p466].

***textTrack*.kind[p476]**

Returns the text track kind[p466] string.

***textTrack*.label[p476]**

Returns the text track label[p466], 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[p476]**

Returns the text track language[p467] string.

***textTrack*.id[p476]**

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()[p474] method.

For text tracks[p466] corresponding to track[p427] elements, this is the ID of the track[p427] element.

***textTrack*.inBandMetadataTrackDispatchType[p476]**

Returns the text track in-band metadata track dispatch type[p466] string.

***textTrack*.mode[p476] [= *value*]**

Returns the text track mode[p467], represented by a string from the following list:

"disabled[p476]"
The text track disabled[p467] mode.

"hidden[p476]"
The text track hidden[p467] mode.

"showing[p476]"
The text track showing[p467] mode.

Can be set, to change the mode.

***textTrack*.cues[p476]**

Returns the text track list of cues[p467], as a TextTrackCueList[p477] object.

***textTrack*.activeCues[p476]**

Returns the text track cues[p468] from the text track list of cues[p467] that are currently active (i.e. that start before the current playback position[p448] and end after it), as a TextTrackCueList[p477] object.

***textTrack*.addCue[p476](*cue*)**

Adds the given cue to *textTrack*'s text track list of cues[p467].

***textTrack*.removeCue[p477](*cue*)**

Removes the given cue from *textTrack*'s text track list of cues[p467].

The **addTextTrack(*kind*, *label*, *language*)** method of media elements[p430], when invoked, must run the following steps:

1. Create a new text track[p466], and set its text track kind[p466] to *kind*, its text track label[p466] to *label*, its text track language[p467] to *language*, its text track readiness state[p467] to the text track loaded[p467] state, its text track mode[p467] to the text track hidden[p467] mode, and its text track list of cues[p467] to an empty list.

   Initially, the text track list of cues[p467] is not associated with any rules for updating the text track rendering[p467]. When a text track cue[p468] is added to it, the text track list of cues[p467] has its rules permanently set accordingly.

2. Add the new text track[p466] to the media element[p430]'s list of text tracks[p466].

3. Queue a media element task[p432] given the media element[p430] to fire an event named addtrack[p486] at the media element[p430]'s

textTracks[p474] attribute's TextTrackList[p474] object, using TrackEvent[p484], with the track[p484] attribute initialized to the new text track[p466].

4. Return the new text track[p466].

The **kind** getter steps are to return this's kind[p466].

The **label** getter steps are to return this's label[p466].

The **language** getter steps are to return this's language[p467].

The **id** getter steps are to return this's identifier[p467].

The **inBandMetadataTrackDispatchType** getter steps are to return this's in-band metadata track dispatch type[p466].

The **mode** getter steps are to return the string switching on this's mode[p467]:

↪ **The text track disabled[p467] mode**
"**disabled**"

↪ **The text track hidden[p467] mode**
"**hidden**"

↪ **The text track showing[p467] mode**
"**showing**"

The mode[p476] setter steps are to set this's mode[p467] to the mode switching on the given value:

↪ "**disabled**[p476]"
The text track disabled[p467] mode.

↪ "**hidden**[p476]"
The text track hidden[p467] mode.

↪ "**showing**[p476]"
The text track showing[p467] mode.

The **cues** getter steps are:

1. If this's mode[p467] is the text track disabled[p467] mode, then return null.
2. Return a live[p48] TextTrackCueList[p477] object that represents the subset of this's text track list of cues[p467] whose end times[p468] occur at or after the earliest possible position when the script started[p476], in text track cue order[p469].

For each TextTrack[p474] object, when an object is returned, the same TextTrackCueList[p477] object must be returned each time.

The **earliest possible position when the script started** is whatever the earliest possible position[p449] was the last time the event loop[p1193] reached step 1.

The **activeCues** getter steps are:

1. If this's mode[p467] is the text track disabled[p467] mode, then return null.
2. Return a live[p48] TextTrackCueList[p477] object that represents the subset of this's text track list of cues[p467] whose active flag was set when the script started[p476], in text track cue order[p469].

For each TextTrack[p474] object, when an object is returned, the same TextTrackCueList[p477] object must be returned each time.

A text track cue[p468]'s **active flag was set when the script started** if its text track cue active flag[p469] was set the last time the event loop[p1193] reached step 1[p1196].

The **addCue(*cue*)** method steps are:

1. Let *list* be this's text track list of cues[p467].

2. If *list* does not yet have any associated rules for updating the text track rendering[p467], then associate *list* with the rules for updating the text track rendering[p467] appropriate to *cue*.

3. If *list*'s associated rules for updating the text track rendering[p467] are not the same rules for updating the text track rendering[p467] as appropriate for *cue*, then throw an "InvalidStateError" DOMException.

4. If the given *cue* is in a text track list of cues[p467], then remove *cue* from that text track list of cues[p467].

5. Add *cue* to *list*.

The **removeCue(*cue*)** method steps are:

1. If the given *cue* is not in this's text track list of cues[p467], then throw a "NotFoundError" DOMException.

2. Remove *cue* from this's text track list of cues[p467].

Example

In this example, an audio[p425] element is used to play a specific sound-effect from a sound file containing many sound effects. A cue is used to pause the audio, so that it ends exactly at the end of the clip, even if the browser is busy running some script. If the page had relied on script to pause the audio, then the start of the next clip might be heard if the browser was not able to run the script at the exact time specified.

```
var sfx = new Audio('sfx.wav');
var sounds = sfx.addTextTrack('metadata');

// add sounds we care about
function addFX(start, end, name) {
  var cue = new VTTCue(start, end, '');
  cue.id = name;
  cue.pauseOnExit = true;
  sounds.addCue(cue);
}
addFX(12.783, 13.612, 'dog bark');
addFX(13.612, 15.091, 'kitten mew');

function playSound(id) {
  sfx.currentTime = sounds.getCueById(id).startTime;
  sfx.play();
}

// play a bark as soon as we can
sfx.oncanplaythrough = function () {
  playSound('dog bark');
}
// meow when the user tries to leave,
// 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**[p478]

Returns the number of cues[p468] in the list.

***cuelist*[*index*]**

Returns the text track cue[p468] with index *index* in the list. The cues are sorted in text track cue order[p469].

***cuelist*.getCueById(*id*)**[p478]

Returns the first text track cue[p468] (in text track cue order[p469]) with text track cue identifier[p468] *id*.

Returns null if none of the cues have the given identifier or if the argument is the empty string.

A TextTrackCueList[p477] object represents a dynamically updating list of text track cues[p468] in a given order.

The **length** attribute must return the number of cues[p468] in the list represented by the TextTrackCueList[p477] object.

The supported property indices of a TextTrackCueList[p477] object at any instant are the numbers from zero to the number of cues[p468] in the list represented by the TextTrackCueList[p477] object minus one, if any. If there are no cues[p468] in the list, there are no supported property indices.

To determine the value of an indexed property for a given index *index*, the user agent must return the *index*th text track cue[p468] in the list represented by the TextTrackCueList[p477] object.

The **getCueById(*id*)** method, when called with an argument other than the empty string, must return the first text track cue[p468] in the list represented by the TextTrackCueList[p477] object whose text track cue identifier[p468] is *id*, if any, or null otherwise. If the argument is the empty string, then the method must return null.

IDL

```
[Exposed=Window]
interface TextTrackCue : EventTarget {
  readonly attribute TextTrack? track;

  attribute DOMString id;
  attribute double startTime;
  attribute unrestricted double endTime;
  attribute boolean pauseOnExit;

  attribute EventHandler onenter;
  attribute EventHandler onexit;
};
```

For web developers (non-normative)

***cue*.track**[p479]

Returns the TextTrack[p474] object to which this text track cue[p468] belongs, if any, or null otherwise.

***cue*.id**[p479] **[= *value*]**

Returns the text track cue identifier[p468].

Can be set.

***cue*.startTime**[p479] **[= *value*]**

Returns the text track cue start time[p468], in seconds.

Can be set.

***cue*.endTime**[p479] **[= *value*]**

Returns the text track cue end time[p468], in seconds.

Returns positive Infinity for an unbounded text track cue[p469].

Can be set.

***cue*.pauseOnExit**[p479] **[= *value*]**

Returns true if the text track cue pause-on-exit flag[p468] is set, false otherwise.

Can be set.

The **track** getter steps are to return the text track[p466] in whose list of cues[p467] this's text track cue[p468] finds itself, if any; or null otherwise.

The **id** attribute, on getting, must return the text track cue identifier[p468] of the text track cue[p468] that the TextTrackCue[p478] object represents. On setting, the text track cue identifier[p468] must be set to the new value.

The **startTime** attribute, on getting, must return the text track cue start time[p468] of the text track cue[p468] that the TextTrackCue[p478] object represents, in seconds. On setting, the text track cue start time[p468] must be set to the new value, interpreted in seconds; then, if the TextTrackCue[p478] object's text track cue[p468] is in a text track[p466]'s list of cues[p467], and that text track[p466] is in a media element[p430]'s list of text tracks[p466], and the media element[p430]'s show poster flag[p449] is not set, then run the *time marches on*[p458] steps for that media element[p430].

The **endTime** attribute, on getting, must return the text track cue end time[p468] of the text track cue[p468] that the TextTrackCue[p478] object represents, in seconds or positive Infinity. On setting, if the new value is negative Infinity or a Not-a-Number (NaN) value, then throw a TypeError exception. Otherwise, the text track cue end time[p468] must be set to the new value. Then, if the TextTrackCue[p478] object's text track cue[p468] is in a text track[p466]'s list of cues[p467], and that text track[p466] is in a media element[p430]'s list of text tracks[p466], and the media element[p430]'s show poster flag[p449] is not set, then run the *time marches on*[p458] steps for that media element[p430].

The **pauseOnExit** attribute, on getting, must return true if the text track cue pause-on-exit flag[p468] of the text track cue[p468] that the TextTrackCue[p478] object represents is set; or false otherwise. On setting, the text track cue pause-on-exit flag[p468] must be set if the new value is true, and must be unset otherwise.

4.8.11.11.6 Event handlers for objects of the text track APIs §[p479]

The following are the event handlers[p1206] that (and their corresponding event handler event types[p1209]) that must be supported, as event handler IDL attributes[p1207], by all objects implementing the TextTrackList[p474] interface:

Event handler[p1206]	Event handler event type[p1209]
onchange	change[p486]
onaddtrack	addtrack[p486]
onremovetrack	removetrack[p486]

The following are the event handlers[p1206] that (and their corresponding event handler event types[p1209]) that must be supported, as event handler IDL attributes[p1207], by all objects implementing the TextTrack[p474] interface:

Event handler[p1206]	Event handler event type[p1209]
oncuechange	cuechange[p486]

The following are the event handlers[p1206] (and their corresponding event handler event types[p1209]) that must be supported, as event handler IDL attributes[p1207], by all objects implementing the TextTrackCue[p478] interface:

Event handler[p1206]	Event handler event type[p1209]
onenter	enter[p486]
onexit	exit[p486]

4.8.11.11.7 Best practices for metadata text tracks §[p479]

This section is non-normative.

Text tracks can be used for storing data relating to the media data, for interactive or augmented views.

For example, a page showing a sports broadcast could include information about the current score. Suppose a robotics competition was being streamed live. The image could be overlaid with the scores, as follows:

In order to make the score display render correctly whenever the user seeks to an arbitrary point in the video, the metadata text track cues need to be as long as is appropriate for the score. For example, in the frame above, there would be maybe one cue that lasts the length of the match that gives the match number, one cue that lasts until the blue alliance's score changes, and one cue that lasts until the red alliance's score changes. If the video is just a stream of the live event, the time in the bottom right would presumably be automatically derived from the current video time, rather than based on a cue. However, if the video was just the highlights, then that might be given in cues also.

The following shows what fragments of this could look like in a WebVTT file:

```
WEBVTT

...

05:10:00.000 --> 05:12:15.000
matchtype:qual
matchnumber:37

...

05:11:02.251 --> 05:11:17.198
red:78

05:11:03.672 --> 05:11:54.198
blue:66

05:11:17.198 --> 05:11:25.912
red:80

05:11:25.912 --> 05:11:26.522
red:83

05:11:26.522 --> 05:11:26.982
red:86

05:11:26.982 --> 05:11:27.499
red:89
```

. . .

The key here is to notice that the information is given in cues that span the length of time to which the relevant event applies. If, instead, the scores were given as zero-length (or very brief, nearly zero-length) cues when the score changes, for example saying "red+2" at 05:11:17.198, "red+3" at 05:11:25.912, etc, problems arise: primarily, seeking is much harder to implement, as the script has to walk the entire list of cues to make sure that no notifications have been missed; but also, if the cues are short it's possible the script will never see that they are active unless it listens to to them specifically.

When using cues in this manner, authors are encouraged to use the `cuechange`[p486] event to update the current annotations. (In particular, using the `timeupdate`[p485] 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`[p485] events are rate-limited.)

4.8.11.12 Identifying a track kind through a URL [§](p48)

Other specifications or formats that need a URL to identify the return values of the `AudioTrack`[p462] `kind`[p464] or `VideoTrack`[p462] `kind`[p464] IDL attributes, or identify the kind of text track[p466], must use the `about:html-kind`[p100] URL.

4.8.11.13 User interface [§](p48)

The **`controls`** attribute is a boolean attribute[p79]. 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[p1152] for the media element[p430], 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[p430]'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[p481] as described above (as if the `controls`[p481] attribute was present).

If the user agent exposes a user interface to the user[p481] by displaying controls over the media element[p430], 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[p460] to the requested position in the media element[p430]'s media timeline[p447]. 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.

For web developers (non-normative)

***media*.volume**[p482] **[= *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[p482] [= *value*]**

Returns true if audio is muted, overriding the volume[p482] attribute, and false if the volume[p482] attribute is being honored.

Can be set, to change whether the audio is muted or not.

A media element[p430] 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.

To **set the playback volume** of a media element[p430] *element* to a number *value*:

1. If *element*'s playback volume[p482] equals *value*, then return.
2. Set *element*'s playback volume[p482] to *value*.
3. If *element* is not allowed to play[p453], then run the internal pause steps[p456] for *element*.
4. Queue a media element task[p432] given *element* to fire an event named volumechange[p486] at *element*.

The **volume** getter steps are to return this's playback volume[p482].

The volume[p482] setter steps are:

1. If the given value is not in the range 0.0 to 1.0 inclusive, then throw an "IndexSizeError" DOMException.
2. Set the playback volume[p482] of this to the given value.

A media element[p430] is **muted** if any of the following are true:

- Its muted state[p482] is true.
- Its muted state[p482] is "default" and it has a muted[p483] content attribute.
- The direction of playback[p457] is backwards.
- Its playbackRate[p454] is so low or so high that the user agent cannot play audio usefully.

Each media element[p430] has a **muted state**, which is either true, false, or "default"; it is initially "default". User agents may set the muted state[p482] of a media element[p430] to true or false (e.g., remembering the last set value across sessions, on a per-site basis or otherwise).

To **set the muted state** of a media element[p430] *element* to a boolean *value*:

1. If *element*'s muted state[p482] equals *value*, then return.
2. Set *element*'s muted state[p482] to *value*.
3. If *element* is not allowed to play[p453], then run the internal pause steps[p456] for *element*.
4. Queue a media element task[p432] given *element* to fire an event named volumechange[p486] at *element*.

The **muted** getter steps are to return true if this is muted[p482]; otherwise false.

The muted[p482] setter steps are to set the muted state[p482] of this to the given value.

A user agent has an associated **volume locked** (a boolean). Its value is implementation-defined and determines whether the playback volume[p482] takes effect.

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 user agent's volume locked[p482] is true, then return the system volume.
3. If the element is muted[p482], then return 0.

4. Let *volume* be the playback volume[p482] of the audio portions of the media element[p430], in range 0.0 (silent) to 1.0 (loudest).

5. 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[p430] is a boolean attribute[p79] that gives the default value for muting.

Note

This attribute has no further effect once the muted[p482] setter has been invoked or the user indicated a preference.

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.

```
<video src="adverts.cgi?kind=video" controls autoplay loop muted></video>
```

4.8.11.14 Time ranges §[p483]

Objects implementing the TimeRanges[p483] 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[p483]**

Returns the number of ranges in the object.

***time* = *media*.start[p483](*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[p483](*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.

A TimeRanges[p483] object has **ranges**, a list of zero or more time ranges[p483].

A **time range** is a struct with the following items:

- A **start**, a number, in seconds.
- An **end**, a number, in seconds.

The **length** getter steps are to return this's ranges[p483]'s size.

The **start(*index*)** method steps are:

1. If *index* is greater than or equal to this's ranges[p483]'s size, then throw an "IndexSizeError" DOMException.
2. Return this's ranges[p483][*index*]'s start[p483].

The **end(*index*)** method steps are:

1. If *index* is greater than or equal to this's ranges[p483]'s size, then throw an "IndexSizeError" DOMException.

2. Return this's ranges[p483][*index*]'s end[p483].

A TimeRanges[p483] object is a **normalized TimeRanges object** if the following are true for each time range[p483] in its ranges[p483]:

- Its start[p483] is greater than the end[p483] of all earlier time ranges[p483].
- Its start[p483] is less than or equal to its end[p483].

In other words, the time ranges[p483] in such an object are ordered, don't overlap, and don't touch (adjacent ones are folded into one bigger time range[p483]). A time range[p483] 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[p432] except for the current frame, when a media element[p430] is paused.

A time range[p483]'s start[p483] and end[p483] are inclusive.

Example

Thus, the end[p483] of a time range[p483] would be equal to the start[p483] of a following adjacent (touching but not overlapping) time range[p483]. Similarly, a time range[p483] covering a whole timeline anchored at zero would have a start[p483] equal to zero and an end[p483] equal to the duration of the timeline.

The timelines used by the objects returned by the buffered[p447], seekable[p461], and played[p455] IDL attributes of media elements[p430] must be that element's media timeline[p447].

4.8.11.15 The TrackEvent[p484] interface §[p484]

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[p484]**

Returns the track object (TextTrack[p474], AudioTrack[p462], or VideoTrack[p462]) to which the event relates.

The **track** attribute must return the value it was initialized to. It represents the context information for the event.

4.8.11.16 Events summary §[p484]

This section is non-normative.

The following events fire on media elements[p430] as part of the processing model described above:

Event name	Interface	Fired when...	Preconditions
loadstart	Event	The user agent begins looking for media data[p431], as part of the resource selection algorithm[p436].	networkState[p435] equals NETWORK_LOADING[p435]
progress	Event	The user agent is fetching media data[p431].	networkState[p435] equals NETWORK_LOADING[p435]
suspend	Event	The user agent is intentionally not currently fetching media data[p431].	networkState[p435] equals NETWORK_IDLE[p435]
abort	Event	The user agent stops fetching the media data[p431] before it is completely downloaded, but not due to an error.	error[p432] is an object with the code MEDIA_ERR_ABORTED[p432]. networkState[p435] equals either NETWORK_EMPTY[p435] or NETWORK_IDLE[p435], depending on when the download was aborted.

Event name	Interface	Fired when...	Preconditions
error	Event	An error occurs while fetching the media data[p431] or the type of the resource is not a supported media format.	error[p432] is an object with the code MEDIA_ERR_NETWORK[p432] or higher. networkState[p435] equals either NETWORK_EMPTY[p435] or NETWORK_IDLE[p435], depending on when the download was aborted.
emptied	Event	A media element[p430] whose networkState[p435] was previously not in the NETWORK_EMPTY[p435] state has just switched to that state (either because of a fatal error during load that's about to be reported, or because the load()[p435] method was invoked while the resource selection algorithm[p436] was already running).	networkState[p435] is NETWORK_EMPTY[p435]; all the IDL attributes are in their initial states.
stalled	Event	The user agent is trying to fetch media data[p431], but data is unexpectedly not forthcoming.	networkState[p435] is NETWORK_LOADING[p435].
loadedmetadata	Event	The user agent has just determined the duration and dimensions of the media resource[p432] and the text tracks are ready[p468].	readyState[p452] is newly equal to HAVE_METADATA[p450] or greater for the first time.
loadeddata	Event	The user agent can render the media data[p431] at the current playback position[p448] for the first time.	readyState[p452] newly increased to HAVE_CURRENT_DATA[p450] or greater for the first time.
canplay	Event	The user agent can resume playback of the media data[p431], but estimates that if playback were to be started now, the media resource[p432] could not be rendered at the current playback rate up to its end without having to stop for further buffering of content.	readyState[p452] newly increased to HAVE_FUTURE_DATA[p450] or greater.
canplaythrough	Event	The user agent estimates that if playback were to be started now, the media resource[p432] could be rendered at the current playback rate all the way to its end without having to stop for further buffering.	readyState[p452] is newly equal to HAVE_ENOUGH_DATA[p450].
playing	Event	Playback is ready to start after having been paused or delayed due to lack of media data[p431].	readyState[p452] is newly greater than or equal to HAVE_FUTURE_DATA[p450] and paused[p453] is false, or paused[p453] is newly false and readyState[p452] is greater than or equal to HAVE_FUTURE_DATA[p450]. Even if this event fires, the element might still not be potentially playing[p453], e.g. if the element is paused for user interaction[p454] or paused for in-band content[p454].
waiting	Event	Playback has stopped because the next frame is not available, but the user agent expects that frame to become available in due course.	readyState[p452] is less than or equal to HAVE_CURRENT_DATA[p450], and paused[p453] is false. Either seeking[p460] is true, or the current playback position[p448] is not contained in any of the ranges in buffered[p447]. It is possible for playback to stop for other reasons without paused[p453] being false, but those reasons do not fire this event (and when those situations resolve, a separate playing[p485] event is not fired either): e.g., playback has ended[p453], or playback stopped due to errors[p453], or the element has paused for user interaction[p454] or paused for in-band content[p454].
seeking	Event	The seeking[p460] IDL attribute changed to true, and the user agent has started seeking to a new position.	
seeked	Event	The seeking[p460] IDL attribute changed to false after the current playback position[p448] was changed.	
ended	Event	Playback has stopped because the end of the media resource[p432] was reached.	currentTime[p449] equals the end of the media resource[p432]; ended[p453] is true.
durationchange	Event	The duration[p449] attribute has just been updated.	
timeupdate	Event	The current playback position[p448] changed as part of normal playback or in an especially interesting way, for example discontinuously.	
play	Event	The element is no longer paused. Fired after the play()[p455] method has returned, or when the autoplay[p452] attribute has caused playback to begin.	paused[p453] is newly false.
pause	Event	The element has been paused. Fired after the pause()[p456] method has returned.	paused[p453] is newly true.
ratechange	Event	Either the defaultPlaybackRate[p454] or the playbackRate[p454] attribute has just been updated.	
resize	Event	One or both of the videoWidth[p424] and	Media element[p430] is a video[p420] element; readyState[p452] is not HAVE_NOTHING[p450]

Event name	Interface	Fired when...	Preconditions
		videoHeight[p424] attributes have just been updated.	
volumechange	Event	Either the volume[p482] attribute or the muted[p482] attribute has changed. Fired after the relevant attribute's setter has returned.	

The following event fires on source[p355] elements:

Event name	Interface	Fired when...
error	Event	An error occurs while fetching the media data[p431] or the type of the resource is not a supported media format.

The following events fire on AudioTrackList[p462], VideoTrackList[p462], and TextTrackList[p474] objects:

Event name	Interface	Fired when...
change	Event	One or more tracks in the track list have been enabled or disabled.
addtrack	TrackEvent[p484]	A track has been added to the track list.
removetrack	TrackEvent[p484]	A track has been removed from the track list.

The following event fires on TextTrack[p474] objects and track[p427] elements:

Event name	Interface	Fired when...
cuechange	Event	One or more cues in the track have become active or stopped being active.

The following events fire on track[p427] elements:

Event name	Interface	Fired when...
error	Event	An error occurs while fetching the track data or the type of the resource is not supported text track format.
load	Event	A track data has been fetched and successfully processed.

The following events fire on TextTrackCue[p478] objects:

Event name	Interface	Fired when...
enter	Event	The cue has become active.
exit	Event	The cue has stopped being active.

4.8.11.17 Security and privacy considerations [§ p486]

The main security and privacy implications of the video[p420] and audio[p425] 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[p135] 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 traversable[p1037].

Example

For instance, if an SVG animation was embedded in a video[p420] 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 traversable with no parent.

If a hostile page embeds victim content, the threat is that the embedding page could obtain information from the content that it would not otherwise have access to. The API does expose some information: the existence of the media, its type, its duration, its size, and the performance characteristics of its host. Such information is already potentially problematic, but in practice the same information can more or less be obtained using the img[p359] 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. [FETCH]

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.

4.8.11.18 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 setting the element's src attribute to an empty string. In cases where srcObject was set, instead set the srcObject to null.

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.

4.8.11.19 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.)

4.8.12 The map element

Categories:

Flow content.

Phrasing content[p157].
Palpable content[p158].

Contexts in which this element can be used[p154]:

Where phrasing content[p157] is expected.

Content model[p154]:

Transparent[p159].

Tag omission in text/html[p154]:

Neither tag is omissible.

Content attributes[p154]:

Global attributes[p162]
name[p488] — Name of image map[p491] to reference[p148] from the usemap[p491] attribute

Accessibility considerations[p154]:

For authors.
For implementers.

Sanitization[p154]:

Uncategorized[p1249].

DOM interface[p154]:

IDL

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

  [CEReactions, Reflect] attribute DOMString name;
  [SameObject] readonly attribute HTMLCollection areas;
};
```

The map[p487] element, in conjunction with an img[p359] element and any area[p489] element descendants, defines an image map[p491]. The element represents[p148] its children.

The **name** attribute gives the map a name so that it can be referenced[p148]. The attribute must be present and must have a non-empty value with no ASCII whitespace. The value of the name[p488] attribute must not be equal to the value of the name[p488] attribute of another map[p487] element in the same tree. If the id[p162] attribute is also specified, both attributes must have the same value.

For web developers (non-normative)

***map*.areas[p488]**

Returns an HTMLCollection of the area[p489] elements in the map[p487].

The **areas** attribute must return an HTMLCollection rooted at the map[p487] element, whose filter matches only area[p489] elements.

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">
<TITLE>Babies™: Toys</TITLE>
<HEADER>
 <H1>Toys</H1>
 <IMG SRC="/images/menu.gif"
      ALT="Babies™ navigation menu. Select a department to go to its page."
      USEMAP="#NAV">
</HEADER>
 ...
<FOOTER>
 <MAP NAME="NAV">
```

```
  <P>
   <A HREF="/clothes/">Clothes</A>
   <AREA ALT="Clothes" COORDS="0,0,100,50" HREF="/clothes/"> |
   <A HREF="/toys/">Toys</A>
   <AREA ALT="Toys" COORDS="100,0,200,50" HREF="/toys/"> |
   <A HREF="/food/">Food</A>
   <AREA ALT="Food" COORDS="200,0,300,50" HREF="/food/"> |
   <A HREF="/books/">Books</A>
   <AREA ALT="Books" COORDS="300,0,400,50" HREF="/books/">
  </P>
 </MAP>
</FOOTER>
```

4.8.13 The area element §[p489]

Categories[p154]:

Flow content[p157].
Phrasing content[p157].

Contexts in which this element can be used[p154]:

Where phrasing content[p157] is expected, but only if there is a map[p487] element ancestor.

Content model[p154]:

Nothing[p155].

Tag omission in text/html[p154]:

No end tag[p1359].

Content attributes[p154]:

Global attributes[p162]
alt[p490] — Replacement text for use when images are not available
coords[p490] — Coordinates for the shape to be created in an image map[p491]
shape[p490] — The kind of shape to be created in an image map[p491]
href[p314] — Address of the hyperlink[p313]
target[p314] — Navigable[p1035] for hyperlink[p313] navigation[p1063]
download[p314] — Whether to download the resource instead of navigating to it, and its filename if so
ping[p314] — URLs to ping
rel[p314] — Relationship between the location in the document containing the hyperlink[p313] and the destination resource
referrerpolicy[p314] — Referrer policy for fetches initiated by the element
hreflang[p316] — Language of the linked resource
type[p316] — Hint for the type of the referenced resource

Accessibility considerations[p154]:

If the element has an href[p314] attribute: for authors; for implementers.
Otherwise: for authors; for implementers.

Sanitization[p154]:

Uncategorized[p1249] with navigating URL attributes[p1251] href[p314].

DOM interface[p154]:

IDL

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

  [CEReactions, Reflect] attribute DOMString alt;
  [CEReactions, Reflect] attribute DOMString coords;
  [CEReactions, Reflect] attribute DOMString shape;
  [CEReactions, Reflect] attribute DOMString download;
```

```
  [CEReactions, Reflect] attribute USVString ping;
  [CEReactions, Reflect] attribute DOMString rel;
  [SameObject, PutForwards=value, Reflect="rel"] readonly attribute DOMTokenList relList;
  [CEReactions] attribute DOMString referrerPolicy;

  // also has obsolete members
};
HTMLAreaElement includes HyperlinkElementUtils;
HTMLAreaElement includes HTMLHyperlinkElementUtils;
```

The area element represents either a hyperlink with some text and a corresponding area on an image map, or a dead area on an image map.

An area element with a parent node must have a map element ancestor.

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 must be omitted.

In both cases, the shape and coords attributes specify the area.

The **shape** attribute is an enumerated attribute with the following keywords and states:

Keyword	Conforming	State	Brief description
circle		Circle state	Designates a circle, using exactly three integers in the coords attribute.
circ	No		
default		Default state	This area is the whole image. (The coords attribute is not used.)
poly		Polygon state	Designates a polygon, using at-least six integers in the coords attribute.
polygon	No		
rect		Rectangle state	Designates a rectangle, using exactly four integers in the coords attribute.
rectangle	No		

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

The **coords** attribute must, if specified, contain a valid list of floating-point numbers. This attribute gives the coordinates for the shape described by the shape attribute. The processing for this attribute is described as part of the image map processing model.

In the **circle state**, area elements must have a coords attribute present, with three integers, the last of which must be non-negative. The first integer must be the distance in CSS pixels from the left edge of the image to the center of the circle, the second integer must be the distance in CSS pixels from the top edge of the image to the center of the circle, and the third integer must be the radius of the circle, again in CSS pixels.

In the **default state**, 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, 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[p314], download[p314], ping[p314], rel[p314], referrerpolicy[p314], hreflang[p316], and type[p316] attributes must be omitted if the href[p314] attribute is not present.

If the itemprop[p833] attribute is specified on an area[p489] element, then the href[p314] attribute must also be specified.

The IDL attribute **referrerPolicy** must reflect[p108] the referrerpolicy[p314] content attribute, limited to only known values[p109].

4.8.14 Image maps §[p49]

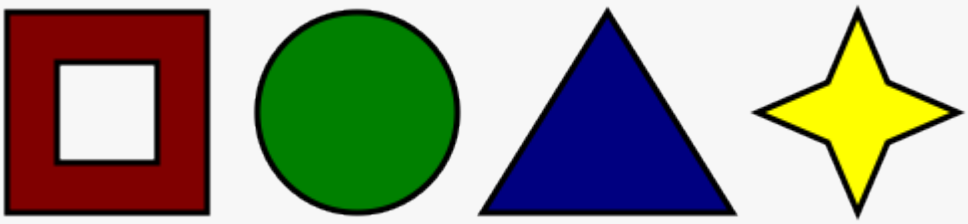
4.8.14.1 Authoring §[p49]

An **image map** allows geometric areas on an image to be associated with hyperlinks[p313].

An image, in the form of an img[p359] element, may be associated with an image map (in the form of a map[p487] element) by specifying a **usemap** attribute on the img[p359] element. The usemap[p491] attribute, if specified, must be a valid hash-name reference[p99] to a map[p487] 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:

```
<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"> <!-- the hole in the red box -->
  <area shape=rect coords="25,25,125,125" href="red.html" alt="Red box.">
  <area shape=circle coords="200,75,50" href="green.html" alt="Green circle.">
  <area shape=poly coords="325,25,262,125,388,125" href="blue.html" alt="Blue triangle.">
  <area shape=poly coords="450,25,435,60,400,75,435,90,450,125,465,90,500,75,465,60"
        href="yellow.html" alt="Yellow star.">
 </map>
</p>
```

4.8.14.2 Processing model §[p49]

If an img[p359] element has a usemap[p491] attribute specified, user agents must process it as follows:

1. Parse the attribute's value using the rules for parsing a hash-name reference[p99] to a map[p487] 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[p489] elements that are descendants of the *map*. Let *areas* be that list.

Having obtained the list of area[p489] 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[p359] element represents, then it must use the following steps:

1. Remove all the area[p489] elements in *areas* that have no href[p314] attribute.
2. Remove all the area[p489] elements in *areas* that have no alt[p490] attribute, or whose alt[p490] attribute's value is the empty string, *if* there is another area[p489] element in *areas* with the same value in the href[p314] attribute and with a non-empty alt[p490] attribute.
3. Each remaining area[p489] element in *areas* represents a hyperlink[p313]. Those hyperlinks should all be made available to the user in a manner associated with the text of the img[p359].

   In this context, user agents may represent area[p489] and img[p359] 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[p489] elements in *areas*, in reverse tree order (so the last specified area[p489] 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[p489] 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[p490] attribute represents.
2. Use the rules for parsing a list of floating-point numbers[p84] to parse the element's coords[p490] attribute, if it is present, and let the *coords* list be the result. 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[p489] element's current state, as per the following table, then the shape is empty; return.

State	Minimum number of items
Circle state[p490]	3
Default state[p490]	0
Polygon state[p490]	6
Rectangle state[p490]	4

4. Check for excess items in the *coords* list as per the entry in the following list corresponding to the shape[p490] attribute's state:

   ↪ **Circle state[p490]**

   Drop any items in the list beyond the third.

   ↪ **Default state[p490]**

   Drop all items in the list.

   ↪ **Polygon state[p490]**

   Drop the last item if there's an odd number of items.

   ↪ **Rectangle state[p490]**

   Drop any items in the list beyond the fourth.

5. If the shape[p490] attribute represents the rectangle state[p490], 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[p490] attribute represents the rectangle state[p490], 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[p490] attribute represents the circle state[p490], 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[p490] attribute:

↪ **Circle state**[p490]

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**[p490]

The shape is a rectangle that exactly covers the entire image.

↪ **Polygon state**[p490]

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][p1582]

↪ **Rectangle state**[p490]

Let x_1 be the first number in *coords*, y_1 be the second number, x_2 be the third number, and y_2 be the fourth number.

The shape is a rectangle whose top-left corner is given by the coordinate (x_1, y_1) and whose bottom right corner is given by the coordinate (x_2, y_2), those coordinates being interpreted as CSS pixels from the top left corner of the image.

For historical reasons, the coordinates must be interpreted relative to the *displayed* image after any stretching caused by the CSS 'width' and 'height' properties (or, for non-CSS browsers, the image element's `width` and `height` attributes — CSS browsers map those attributes to the aforementioned CSS properties).

Note

Browser zoom features and transforms applied using CSS or SVG do not affect the coordinates.

Pointing device interaction with an image associated with a set of layered shapes per the above algorithm must result in the relevant user interaction events being first fired to the top-most shape covering the point that the pointing device indicated, if any, or to the image element itself, if there is no shape covering that point. User agents may also allow individual `area`[p489] elements representing hyperlinks[p313] to be selected and activated (e.g. using a keyboard).

Note

Because a `map`[p487] element (and its `area`[p489] elements) can be associated with multiple `img`[p359] elements, it is possible for an `area`[p489] element to correspond to multiple focusable areas[p874] of the document.

Image maps are live[p48]; if the DOM is mutated, then the user agent must act as if it had rerun the algorithms for image maps.

4.8.15 MathML §[p493]

The MathML `math` element falls into the embedded content[p158], phrasing content[p157], flow content[p157], and palpable content[p158] 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[p157].

When the MathML token elements (`mi`, `mo`, `mn`, `ms`, and `mtext`) are descendants of HTML elements, they may contain phrasing content[p157] elements from the HTML namespace.

User agents must handle text other than inter-element whitespace[p155] 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[p77]. [MATHML][p1584]

Example

Here is an example of the use of MathML in an HTML document:

```
<!DOCTYPE html>
<html lang="en">
 <head>
  <title>The quadratic formula</title>
 </head>
 <body>
  <h1>The quadratic formula</h1>
  <p>
   <math>
    <mi>x</mi>
    <mo>=</mo>
    <mfrac>
     <mrow>
      <mo form="prefix">−</mo> <mi>b</mi>
      <mo>±</mo>
      <msqrt>
       <msup> <mi>b</mi> <mn>2</mn> </msup>
       <mo>−</mo>
       <mn>4</mn> <mo></mo> <mi>a</mi> <mo></mo> <mi>c</mi>
      </msqrt>
     </mrow>
     <mrow>
      <mn>2</mn> <mo></mo> <mi>a</mi>
     </mrow>
    </mfrac>
   </math>
  </p>
 </body>
</html>
```

4.8.16 SVG §[p494]

The SVG `svg` element falls into the embedded content[p158], phrasing content[p157], flow content[p157], and palpable content[p158] 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[p157].

The content model for the SVG `title` element inside HTML documents is phrasing content[p157]. (This further constrains the requirements given in *SVG 2*.)

The semantics of SVG elements are defined by *SVG 2* and other applicable specifications[p77]. [SVG][p1586]

For web developers (non-normative)

***doc* = *iframe*.getSVGDocument[p494]()**

***doc* = *embed*.getSVGDocument[p494]()**

***doc* = *object*.getSVGDocument[p494]()**

Returns the `Document`[p135] object, in the case of `iframe`[p404], `embed`[p413], or `object`[p416] elements being used to embed SVG.

The **`getSVGDocument()`** method steps are:

1. Let *document* be this's content document[p1039].
2. If *document* is non-null and was created by the page load processing model for XML files[p1110] section because the computed

type of the resource in the navigate[p1063] algorithm was image/svg+xml[p1577], then return *document*.

3. Return null.

4.8.17 Dimension attributes §[p495]

Author requirements: The **width** and **height** attributes on img[p359], iframe[p404], embed[p413], object[p416], video[p420], source[p355] when the parent is a picture[p355] element and, when their type[p541] attribute is in the Image Button[p566] state, input[p539] 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[p81].

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 natural width to the natural height in the resource. The *specified width* and *specified height* are the values of the width[p495] and height[p495] attributes respectively.

The two attributes must be omitted if the resource in question does not have both a natural width and a natural height.

If the two attributes are both 0, it indicates that the element is not intended for the user (e.g. it might be a part of a service to count page views).

Note

The dimension attributes are not intended to be used to stretch the image.

User agent requirements: User agents are expected to use these attributes as hints for the rendering[p1510].

Note

For iframe[p404], embed[p413] and object[p416] the IDL attributes are DOMString; for video[p420] and source[p355] the IDL attributes are unsigned long.

Note

The corresponding IDL attributes for img[p364] and input[p545] elements are defined in those respective elements' sections, as they are slightly more specific to those elements' other behaviors. Both are defined in terms of the algorithm below.

To **determine the dimensions** of an img[p359] or input[p539] element *element* given a struct consisting of a width and a height, or null, *naturalDimensions*:

1. If *element* is being rendered[p1488], then return the width and height of *element*'s content box, in CSS pixels. [CSS][p1580]
2. Let *width* and *height* be 0.
3. If *naturalDimensions* is not null, then set *width* to *naturalDimensions*'s width and *height* to *naturalDimensions*'s height.
4. If *element* has a width[p495] attribute:
    1. Let *attributeWidth* be the result of parsing that attribute's value[p81].
    2. If *attributeWidth* is not an error, then set *width* to *attributeWidth*.
5. If *element* has a height[p495] attribute:
    1. Let *attributeHeight* be the result of parsing that attribute's value[p81].
    2. If *attributeHeight* is not an error, then set *height* to *attributeHeight*.

6. Return a width of *width* and a height of *height*, in CSS pixels.

4.9 Tabular data §[p496]

4.9.1 The table element §[p496]

Categories[p154]:

Flow content[p157].
Palpable content[p158].

Contexts in which this element can be used[p154]:

Where flow content[p157] is expected.

Content model[p154]:

In this order: optionally a caption[p504] element, followed by zero or more colgroup[p505] elements, followed optionally by a thead[p508] element, followed by either zero or more tbody[p507] elements or one or more tr[p510] elements, followed optionally by a tfoot[p509] element, optionally intermixed with one or more script-supporting elements[p159].

Tag omission in text/html[p154]:

Neither tag is omissible.

Content attributes[p154]:

Global attributes[p162]

Accessibility considerations[p154]:

For authors.
For implementers.

Sanitization[p154]:

Default[p1249].

DOM interface[p154]:

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
};
```

The table[p496] element represents[p148] data with more than one dimension, in the form of a table[p516].

The table[p496] element takes part in the table model[p516]. Tables have rows, columns, and cells given by their descendants. The rows and columns form a grid; a table's cells must completely cover that grid without overlap.

Note

Precise rules for determining whether this conformance requirement is met are described in the description of the table model[p516].

Authors are encouraged to provide information describing how to interpret complex tables. Guidance on how to provide such information[p501] 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, such as CSS grid layout, CSS flexible box layout ("flexbox"), CSS multi-column layout, CSS positioning, and the CSS table model. [CSS][p1580]

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.

Note

Authors and implementers are encouraged to consider using some of the table design techniques[p504] described below to make tables easier to navigate for users.

User agents, especially those that do table analysis on arbitrary content, are encouraged to find heuristics to determine which tables actually contain data and which are merely being used for layout. This specification does not define a precise heuristic, but the following are suggested as possible indicators:

Feature	Indication
The use of the role[p70] attribute with the value presentation	Probably a layout table
The use of the non-conforming border[p1535] attribute with the non-conforming value 0	Probably a layout table
The use of the non-conforming cellspacing[p1535] and cellpadding[p1535] attributes with the value 0	Probably a layout table
The use of caption[p504], thead[p508], or th[p513] elements	Probably a non-layout table
The use of the headers[p515] and scope[p514] attributes	Probably a non-layout table
The use of the non-conforming border[p1535] attribute with a value other than 0	Probably a non-layout table
Explicit visible borders set using CSS	Probably a non-layout table
The use of the summary[p1533] attribute	Not a good indicator (both layout and non-layout tables have historically been given this attribute)

Note

It is quite possible that the above suggestions are wrong. Implementers are urged to provide feedback elaborating on their experiences with trying to create a layout table detection heuristic.

If a table[p496] element has a (non-conforming) summary[p1533] 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[p498] [= *value*]**

Returns the table's caption[p504] element.

Can be set, to replace the caption[p504] element.

***caption* = *table*.createCaption[p498]()**

Ensures the table has a caption[p504] element, and returns it.

***table*.deleteCaption[p499]()**

Ensures the table does not have a caption[p504] element.

***table*.tHead[p499] [= *value*]**

Returns the table's thead[p508] element.

Can be set, to replace the thead[p508] element. If the new value is not a thead[p508] element, throws a "HierarchyRequestError" DOMException.

***thead* = *table*.createTHead[p499]()**

Ensures the table has a thead[p508] element, and returns it.

***table*.deleteTHead[p499]()**

Ensures the table does not have a thead[p508] element.

***table*.tFoot[p499] [= *value*]**

Returns the table's tfoot[p509] element.

Can be set, to replace the tfoot[p509] element. If the new value is not a tfoot[p509] element, throws a "HierarchyRequestError" DOMException.

***tfoot* = *table*.createTFoot[p499]()**

Ensures the table has a tfoot[p509] element, and returns it.

***table*.deleteTFoot[p499]()**

Ensures the table does not have a tfoot[p509] element.

***table*.tBodies[p499]**

Returns an HTMLCollection of the tbody[p507] elements of the table.

***tbody* = *table*.createTBody[p499]()**

Creates a tbody[p507] element, inserts it into the table, and returns it.

***table*.rows[p499]**

Returns an HTMLCollection of the tr[p510] elements of the table.

***tr* = *table*.insertRow[p500]([*index*])**

Creates a tr[p510] element, along with a tbody[p507] if required, inserts them into the table at the position given by the argument, and returns the tr[p510].

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.

***table*.deleteRow[p500](*index*)**

Removes the tr[p510] element with the given position in the table.

The position is relative to the rows in the table. The index −1 is equivalent to deleting the last row of the table.

If the given position is less than −1 or greater than the index of the last row, or if there are no rows, throws an "IndexSizeError" DOMException.

To **create a table element** given a table[p496] element *tableElement* and a string *localName*, return the result of creating an element given *tableElement*'s node document, *localName*, and the HTML namespace.

The **caption** getter steps are to return the first caption[p504] element child of this, if any; otherwise null.

The caption[p498] setter steps are:

1. Remove the first caption[p504] element child of this, if any.
2. If the given value is not null, then insert it as the first node of this.

The **createCaption()** method steps are:

1. If this has a caption[p504] element child, then return the first such element.
2. Let *caption* be the result of creating a table element[p498] given this and "caption".

3. Insert *caption* as the first node of this.
4. Return *caption*.

The **deleteCaption()** method steps are to remove the first caption element child of this, if any.

The **tHead** getter steps are to return the first thead element child of this, if any; otherwise null.

The tHead setter steps are:

1. If the given value is neither null nor a thead element, then throw a "HierarchyRequestError" DOMException.
2. Remove the first thead element child of this, if any.
3. If the given value is not null, then insert it immediately before the first element child of this that is neither a caption element nor a colgroup element, if any; otherwise insert it at the end of this.

The **createTHead()** method steps are:

1. If this has a thead element child, then return the first such element.
2. Let *thead* be the result of creating a table element given this and "thead".
3. Insert *thead* immediately before the first element child of this that is neither a caption element nor a colgroup element, if any; otherwise insert *thead* at the end of this.
4. Return *thead*.

The **deleteTHead()** method steps are to remove the first thead element child of this, if any.

The **tFoot** getter steps are to return the first tfoot element child of this, if any; otherwise null.

The tFoot setter steps are:

1. If the given value is neither null nor a tfoot element, then throw a "HierarchyRequestError" DOMException.
2. Remove the first tfoot element child of this, if any.
3. If the given value is not null, then insert it at the end of this.

The **createTFoot()** method steps are:

1. If this has a tfoot element child, then return the first such element.
2. Let *tfoot* be the result of creating a table element given this and "tfoot".
3. Insert *tfoot* at the end of this.
4. Return *tfoot*.

The **deleteTFoot()** method steps are to remove the first tfoot element child of this, if any.

The **tBodies** getter steps are to return an HTMLCollection rooted at this, whose filter matches only tbody elements that are children of this.

The **createTBody()** method steps are:

1. Let *tbody* be the result of creating a table element given this and "tbody".
2. Insert *tbody* immediately after the last tbody element child of this, if any; otherwise insert *tbody* at the end of this.
3. Return *tbody*.

The **rows** getter steps are to return an HTMLCollection rooted at this, whose filter matches only tr elements that are either children of this, or children of thead, tbody, or tfoot elements that are themselves children of this. 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 this or a tbody element, again in tree order, followed finally by those elements whose parent is a tfoot element, still in tree order.

The **`insertRow(`*`index`*`)`** method steps are:

1. If *index* is less than −1 or greater than the number of elements in the rows[p499] collection, then throw an "IndexSizeError" DOMException.
2. Let *tr* be the result of creating a table element[p498] given this and "`tr`".
3. If the rows[p499] collection has zero elements in it, and this has no tbody[p507] elements in it:
    1. Let *tbody* be the result of creating a table element[p498] given this and "`tbody`".
    2. Append *tr* to *tbody*.
    3. Append *tbody* to this.
4. Otherwise, if the rows[p499] collection has zero elements in it, then append *tr* to the last tbody[p507] element in this.
5. Otherwise, if *index* is −1 or equal to the number of items in the rows[p499] collection, then append *tr* to the parent of the last tr[p510] element in the rows[p499] collection.
6. Otherwise, insert *tr* immediately before the *index*th tr[p510] element in the rows[p499] collection, in the same parent.
7. Return *tr*.

The **`deleteRow(`*`index`*`)`** method steps are:

1. If *index* is less than −1 or greater than or equal to the number of elements in the rows[p499] collection, then throw an "IndexSizeError" DOMException.
2. If *index* is −1, then remove the last element in the rows[p499] collection from its parent, or do nothing if the rows[p499] collection is empty.
3. Otherwise, remove the *index*th element in the rows[p499] 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.

```
<style>
 #sudoku { border-collapse: collapse; border: solid thick; }
 #sudoku colgroup, 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><col><col><col>
 <colgroup><col><col><col>
 <tbody>
  <tr> <td> 1 <td>   <td> 3 <td> 6 <td>   <td> 4 <td> 7 <td>   <td> 9
  <tr> <td>   <td> 2 <td>   <td>   <td> 9 <td>   <td>   <td> 1 <td>
  <tr> <td> 7 <td>   <td>   <td>   <td>   <td>   <td>   <td>   <td> 6
 <tbody>
  <tr> <td> 2 <td>   <td> 4 <td>   <td> 3 <td>   <td> 9 <td>   <td> 8
  <tr> <td>   <td>   <td>   <td>   <td>   <td>   <td>   <td>   <td>
  <tr> <td> 5 <td>   <td>   <td> 9 <td>   <td> 7 <td>   <td>   <td> 1
 <tbody>
  <tr> <td> 6 <td>   <td>   <td>   <td> 5 <td>   <td>   <td>   <td> 2
  <tr> <td>   <td>   <td>   <td>   <td> 7 <td>   <td>   <td>   <td>
  <tr> <td> 9 <td>   <td>   <td> 8 <td>   <td> 2 <td>   <td>   <td> 5
</table>
```

4.9.1.1 Techniques for describing tables §[p50]

For tables that consist of more than just a grid of cells with headers in the first row and headers in the first column, and for any table in general where the reader might have difficulty understanding the content, authors should include explanatory information introducing the table. This information is useful for all users, but is especially useful for users who cannot see the table, e.g. users of screen readers.

Such explanatory information should introduce the purpose of the table, outline its basic cell structure, highlight any trends or patterns, and generally teach the user how to use the table.

For instance, the following table:

Characteristics with positive and negative sides

Negative	Characteristic	Positive
Sad	**Mood**	Happy
Failing	**Grade**	Passing

...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

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> Characteristic
   <th> Positive
 <tbody>
  <tr>
   <td headers="n r1"> Sad
   <th id="r1"> Mood
   <td> Happy
  <tr>
   <td headers="n r2"> Failing
   <th id="r2"> Grade
   <td> Passing
</table>
```

In the table's caption[p504]

Example

```
<table>
 <caption>
  <strong>Characteristics with positive and negative sides.</strong>
  <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>
 </caption>
 <thead>
  <tr>
   <th id="n"> Negative
   <th> Characteristic
```

```
   <th> Positive
 <tbody>
  <tr>
   <td headers="n r1"> Sad
   <th id="r1"> Mood
   <td> Happy
  <tr>
   <td headers="n r2"> Failing
   <th id="r2"> Grade
   <td> Passing
</table>
```

In the table's caption[p504], in a details[p666] element

Example

```
<table>
 <caption>
  <strong>Characteristics with positive and negative sides.</strong>
  <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>
 </caption>
 <thead>
  <tr>
   <th id="n"> Negative
   <th> Characteristic
   <th> Positive
 <tbody>
  <tr>
   <td headers="n r1"> Sad
   <th id="r1"> Mood
   <td> Happy
  <tr>
   <td headers="n r2"> Failing
   <th id="r2"> Grade
   <td> Passing
</table>
```

Next to the table, in the same figure[p259]

Example

```
<figure>
 <figcaption>Characteristics with positive and negative sides</figcaption>
 <p>Characteristics are given in the second column, with the
 negative side in the left column and the positive side in the right
 column.</p>
 <table>
  <thead>
   <tr>
    <th id="n"> Negative
    <th> Characteristic
    <th> Positive
  <tbody>
   <tr>
    <td headers="n r1"> Sad
```

```
    <th id="r1"> Mood
    <td> Happy
   <tr>
    <td headers="n r2"> Failing
    <th id="r2"> Grade
    <td> Passing
 </table>
</figure>
```

Next to the table, in a figure[p259]'s figcaption[p262]

Example

```
<figure>
 <figcaption>
  <strong>Characteristics with positive and negative sides</strong>
  <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>
 </figcaption>
 <table>
  <thead>
   <tr>
    <th id="n"> Negative
    <th> Characteristic
    <th> Positive
  <tbody>
   <tr>
    <td headers="n r1"> Sad
    <th id="r1"> Mood
    <td> Happy
   <tr>
    <td headers="n r2"> Failing
    <th id="r2"> Grade
    <td> Passing
 </table>
</figure>
```

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[p515] attributes:

```
<table>
 <caption>Characteristics with positive and negative sides</caption>
 <thead>
  <tr>
   <th> Characteristic
   <th> Negative
   <th> Positive
 <tbody>
  <tr>
   <th> Mood
   <td> Sad
   <td> Happy
```

```
  <tr>
   <th> Grade
   <td> Failing
   <td> Passing
</table>
```

4.9.1.2 Techniques for table design §[p504]

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 §[p504]

Categories[p154]:

None.

Contexts in which this element can be used[p154]:

As the first element child of a `table`[p496] element.

Content model[p154]:

Flow content[p157], but with no descendant `table`[p496] elements.

Tag omission in text/html[p154]:

A `caption`[p504] element's end tag[p1359] can be omitted if the `caption`[p504] element is not immediately followed by ASCII whitespace or a comment[p1367].

Content attributes[p154]:

Global attributes[p162]

Accessibility considerations[p154]:

For authors.
For implementers.

Sanitization[p154]:

Default[p1249].

DOM interface[p154]:

IDL

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

  // also has obsolete members
};
```

The `caption`[p504] element represents[p148] the title of the `table`[p496] that is its parent, if it has a parent and that is a `table`[p496] 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:

	1	2	3	4	5	6
1	2	3	4	5	6	7
2	3	4	5	6	7	8
3	4	5	6	7	8	9
4	5	6	7	8	9	10
5	6	7	8	9	10	11
6	7	8	9	10	11	12

In the abstract, this table is not clear. However, with a caption giving the table's number (for reference in the main prose) and explaining its use, it makes more sense:

```
<caption>
<p>Table 1.
<p>This table shows the total score obtained from rolling two
six-sided dice. The first row represents the value of the first die,
the first column the value of the second die. The total is given in
the cell that corresponds to the values of the two dice.
</caption>
```

This provides the user with more context:

Table 1.

This table shows the total score obtained from rolling two six-sided dice. The first row represents the value of the first die, the first column the value of the second die. The total is given in the cell that corresponds to the values of the two dice.

	1	2	3	4	5	6
1	2	3	4	5	6	7
2	3	4	5	6	7	8
3	4	5	6	7	8	9
4	5	6	7	8	9	10
5	6	7	8	9	10	11
6	7	8	9	10	11	12

4.9.3 The colgroup element

Categories:

None.

Contexts in which this element can be used:

As a child of a table element, after any caption elements and before any thead, tbody, tfoot, and tr elements.

Content model:

If the span attribute is present: Nothing.
If the span attribute is absent: Zero or more col and template elements.

Tag omission in text/html:

A colgroup element's start tag can 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 can be omitted if the colgroup element is not immediately followed by ASCII whitespace or a comment.

Content attributes:

Global attributes
span — Number of columns spanned by the element

Accessibility considerations:

For authors.
For implementers.

Sanitization:

Default with attributes span.

DOM interface:

IDL
```
[Exposed=Window]
interface HTMLTableColElement : HTMLElement {
  [HTMLConstructor] constructor();

  [CEReactions, Reflect, ReflectDefault=1, ReflectRange=(1, 1000)] attribute unsigned long span;

  // also has obsolete members
};
```

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.

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.

Sanitization:

Default with attributes span.

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.

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.

Sanitization:

Default.

DOM interface:

IDL

```
[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**

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 §

Categories:

None.

Contexts in which this element can be used:

As a child of a table element, after any caption, and colgroup elements and before any tbody, tfoot, and tr elements, but only if there are no other thead elements that are children of the table element.

Content model:

Zero or more tr and script-supporting elements.

Tag omission in text/html:

A thead element's end tag 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.

Sanitization[p154]**:**

Default[p1249].

DOM interface[p154]**:**

Uses HTMLTableSectionElement[p507], as defined for tbody[p507] elements.

The thead[p508] element represents[p148] the block[p516] of rows[p516] that consist of the column labels (headers) and any ancillary non-header cells for the parent table[p496] element, if the thead[p508] element has a parent and it is a table[p496].

The thead[p508] element takes part in the table model[p516].

Example

This example shows a thead[p508] element being used. Notice the use of both th[p513] and td[p512] elements in the thead[p508] element: the first row is the headers, and the second row is an explanation of how to fill in the table.

```
<table>
 <caption> School auction sign-up sheet </caption>
 <thead>
  <tr>
   <th><label for=e1>Name</label>
   <th><label for=e2>Product</label>
   <th><label for=e3>Picture</label>
   <th><label for=e4>Price</label>
  <tr>
   <td>Your name here
   <td>What are you selling?
   <td>Link to a picture
   <td>Your reserve price
 <tbody>
  <tr>
   <td>Ms Danus
   <td>Doughnuts
   <td><img src="https://example.com/mydoughnuts.png" title="Doughnuts from Ms Danus">
   <td>$45
  <tr>
   <td><input id=e1 type=text name=who required form=f>
   <td><input id=e2 type=text name=what required form=f>
   <td><input id=e3 type=url name=pic form=f>
   <td><input id=e4 type=number step=0.01 min=0 value=0 required form=f>
</table>
<form id=f action="/auction.cgi">
 <input type=button name=add value="Submit">
</form>
```

4.9.7 The tfoot element §[p509]

Categories[p154]**:**

None.

Contexts in which this element can be used[p154]**:**

As a child of a table[p496] element, after any caption[p504], colgroup[p505], thead[p508], tbody[p507], and tr[p510] elements, but only if there are no other tfoot[p509] elements that are children of the table[p496] element.

Content model[p154]**:**

Zero or more tr[p510] and script-supporting[p159] elements.

Tag omission in text/html[p154]**:**

A tfoot[p509] element's end tag[p1359] can be omitted if there is no more content in the parent element.

Content attributes[p154]**:**

Global attributes[p162]

Accessibility considerations[p154]**:**

For authors.
For implementers.

Sanitization[p154]**:**

Default[p1249].

DOM interface[p154]**:**

Uses `HTMLTableSectionElement`[p507], as defined for `tbody`[p507] elements.

The `tfoot`[p509] element represents[p148] the block[p516] of rows[p516] that consist of the column summaries (footers) for the parent `table`[p496] element, if the `tfoot`[p509] element has a parent and it is a `table`[p496].

The `tfoot`[p509] element takes part in the table model[p516].

4.9.8 The `tr` element §[p510]

Categories[p154]**:**

None.

Contexts in which this element can be used[p154]**:**

As a child of a `thead`[p508] element.
As a child of a `tbody`[p507] element.
As a child of a `tfoot`[p509] element.
As a child of a `table`[p496] element, after any `caption`[p504], `colgroup`[p505], and `thead`[p508] elements, but only if there are no `tbody`[p507] elements that are children of the `table`[p496] element.

Content model[p154]**:**

Zero or more `td`[p512], `th`[p513], and script-supporting[p159] elements.

Tag omission in text/html[p154]**:**

A `tr`[p510] element's end tag[p1359] can be omitted if the `tr`[p510] element is immediately followed by another `tr`[p510] element, or if there is no more content in the parent element.

Content attributes[p154]**:**

Global attributes[p162]

Accessibility considerations[p154]**:**

For authors.
For implementers.

Sanitization[p154]**:**

Default[p1249].

DOM interface[p154]**:**

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.

For web developers (non-normative)

***tr*.rowIndex**

Returns the position of the row in the table's rows list.

Returns −1 if the element isn't in a table.

***tr*.sectionRowIndex**

Returns the position of the row in the table section's rows list.

Returns −1 if the element isn't in a table section.

***tr*.cells**

Returns an HTMLCollection of the td and th elements of the row.

***cell* = *tr*.insertCell([*index*])**

Creates a td element, inserts it into the table row at the position given by the argument, and returns the td.

The position is relative to the cells in the row. The index −1, which is the default if the argument is omitted, is equivalent to inserting at the end of the row.

If the given position is less than −1 or greater than the number of cells, throws an "IndexSizeError" 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.

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 rooted at this tr element, whose filter matches only td and th elements that are children of the tr element.

The **insertCell(*index*)** method must act as follows:

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[p511] collection from its parent.

4.9.9 The td element §[p512]

Categories[p154]:

None.

Contexts in which this element can be used[p154]:

As a child of a tr[p510] element.

Content model[p154]:

Flow content[p157].

Tag omission in text/html[p154]:

A td[p512] element's end tag[p1359] can be omitted if the td[p512] element is immediately followed by a td[p512] or th[p513] element, or if there is no more content in the parent element.

Content attributes[p154]:

Global attributes[p162]
colspan[p515] — Number of columns that the cell is to span
rowspan[p515] — Number of rows that the cell is to span
headers[p515] — The header cells for this cell

Accessibility considerations[p154]:

For authors.
For implementers.

Sanitization[p154]:

Default[p1249] with attributes colspan[p515], headers[p515], rowspan[p515].

DOM interface[p154]:

IDL

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

  [CEReactions, Reflect, ReflectDefault=1, ReflectRange=(1, 1000)] attribute unsigned long
colSpan;
  [CEReactions, Reflect, ReflectDefault=1, ReflectRange=(0, 65534)] attribute unsigned long
rowSpan;
  [CEReactions, Reflect] attribute DOMString headers;
  readonly attribute long cellIndex;

  [CEReactions] attribute DOMString scope; // only conforming for th elements
  [CEReactions, Reflect] attribute DOMString abbr;  // only conforming for th elements

  // also has obsolete members
};
```

The HTMLTableCellElement[p512] interface is also used for th[p513] elements.

The td[p512] element represents[p148] a data cell[p516] in a table.

The td[p512] element and its colspan[p515], rowspan[p515], and headers[p515] attributes take part in the table model[p516].

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[p516], or listing the cell's header cells (as determined by the algorithm for assigning header cells[p519]). When a cell's header cells are being listed, user agents may use the value of abbr[p514] 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[p513] elements instead of td[p512] elements. A script would attach event handlers to these elements to maintain the total.

```
<table>
 <tr>
  <th><input value="Name">
  <th><input value="Paid ($)">
 <tr>
  <td><input value="Jeff">
  <td><input value="14">
 <tr>
  <td><input value="Britta">
  <td><input value="9">
 <tr>
  <td><input value="Abed">
  <td><input value="25">
 <tr>
  <td><input value="Shirley">
  <td><input value="2">
 <tr>
  <td><input value="Annie">
  <td><input value="5">
 <tr>
  <td><input value="Troy">
  <td><input value="5">
 <tr>
  <td><input value="Pierce">
  <td><input value="1000">
 <tr>
  <th><input value="Total">
  <td><output value="1060">
</table>
```

✓ MDN

4.9.10 The th element §[p513]

Categories[p154]:

None.

Contexts in which this element can be used[p154]:

As a child of a tr[p510] element.

Content model[p154]:

Flow content[p157], but with no header[p226], footer[p227], sectioning content[p157], or heading content[p157] descendants.

Tag omission in text/html[p154]:

A th[p513] element's end tag[p1359] can be omitted if the th[p513] element is immediately followed by a td[p512] or th[p513] element, or if there is no more content in the parent element.

Content attributes[p154]:

Global attributes[p162]
colspan[p515] — Number of columns that the cell is to span
rowspan[p515] — Number of rows that the cell is to span
headers[p515] — The header cells for this cell
scope[p514] — Specifies which cells the header cell applies to
abbr[p514] — Alternative label to use for the header cell when referencing the cell in other contexts

Accessibility considerations[p154]:

For authors.
For implementers.

Sanitization[p154]**:**

Default[p1249] with attributes abbr[p514], colspan[p515], headers[p515], rowspan[p515], scope[p514].

DOM interface[p154]**:**

Uses HTMLTableCellElement[p512], as defined for td[p512] elements.

The th[p513] element represents[p148] a header cell[p516] in a table.

The th[p513] element may have a **scope** content attribute specified.

The scope[p514] attribute is an enumerated attribute[p80] with the following keywords and states:

Keyword	State	Brief description
row	**Row**	The header cell applies to some of the subsequent cells in the same row(s).
col	**Column**	The header cell applies to some of the subsequent cells in the same column(s).
rowgroup	**Row Group**	The header cell applies to all the remaining cells in the row group.
colgroup	**Column Group**	The header cell applies to all the remaining cells in the column group.

The attribute's *missing value default*[p80] and *invalid value default*[p80] are both the **Auto** state. (In this state the header cell applies to a set of cells selected based on context.)

A th[p513] element's scope[p514] attribute must not be in the Row Group[p514] state if the element is not anchored in a row group[p516], nor in the Column Group[p514] state if the element is not anchored in a column group[p516].

The th[p513] 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[p513] element and its colspan[p515], rowspan[p515], headers[p515], and scope[p514] attributes take part in the table model[p516].

Example

The following example shows how the scope[p514] attribute's rowgroup[p514] value affects which data cells a header cell applies to.

Here is a markup fragment showing a table:

```
<table>
 <thead>
  <tr> <th> ID <th> Measurement <th> Average <th> Maximum
 <tbody>
  <tr> <td> <th scope=rowgroup> Cats <td> <td>
  <tr> <td> 93 <th> Legs <td> 3.5 <td> 4
  <tr> <td> 10 <th> Tails <td> 1 <td> 1
 <tbody>
  <tr> <td> <th scope=rowgroup> English speakers <td> <td>
  <tr> <td> 32 <th> Legs <td> 2.67 <td> 4
  <tr> <td> 35 <th> Tails <td> 0.33 <td> 1
</table>
```

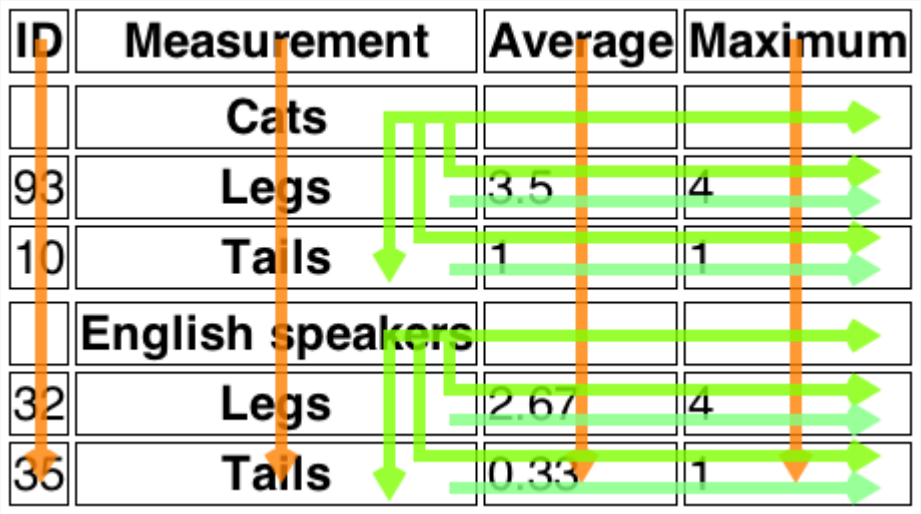
This would result in the following table:

ID	Measurement	Average	Maximum
	Cats		
93	**Legs**	3.5	4
10	**Tails**	1	1
	English speakers		
32	**Legs**	2.67	4
35	**Tails**	0.33	1

The headers in the first row all apply directly down to the rows in their column.

The headers with a scope[p514] attribute in the Row Group[p514] state 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.



4.9.11 Attributes common to td[p512] and th[p513] elements §[p515]

The td[p512] and th[p513] elements may have a **colspan** content attribute specified, whose value must be a valid non-negative integer[p81] greater than zero and less than or equal to 1000.

The td[p512] and th[p513] elements may also have a **rowspan** content attribute specified, whose value must be a valid non-negative integer[p81] 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[p516].

The td[p512] and th[p513] element may have a **headers** content attribute specified. The headers[p515] attribute, if specified, must contain a string consisting of an unordered set of unique space-separated tokens[p99], none of which are identical to another token and each of which must have the value of an ID of a th[p513] element taking part in the same table[p516] as the td[p512] or th[p513] element (as defined by the table model[p516]).

A th[p513] element with ID *id* is said to be *directly targeted* by all td[p512] and th[p513] elements in the same table[p516] that have headers[p515] attributes whose values include as one of their tokens the ID *id*. A th[p513] element *A* is said to be *targeted* by a th[p513] or td[p512] 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[p513] element must not be *targeted* by itself.

The colspan[p515], rowspan[p515], and headers[p515] attributes take part in the table model[p516].

For web developers (non-normative)

***cell*.cellIndex**[p515]

Returns the position of the cell in the row's cells[p511] 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 **cellIndex** IDL attribute must, if the element has a parent tr[p510] element, return the index of the cell's element in the parent

element's cells[p511] collection. If there is no such parent element, then the attribute must return −1.

The **scope** IDL attribute must reflect[p108] the content attribute of the same name, limited to only known values[p109].

4.9.12 Processing model §[p516]

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 $0 \le x < x_{width}$, and the y coordinates are always in the range $0 \le y < y_{height}$. If one or both of x_{width} and y_{height} are zero, then the table is empty (has no slots). Tables correspond to table[p496] elements.

A **cell** is a set of slots anchored at a slot ($cell_x$, $cell_y$), and with a particular *width* and *height* such that the cell covers all the slots with coordinates (x, y) where $cell_x \le x < cell_x+width$ and $cell_y \le y < cell_y+height$. Cells can either be *data cells* or *header cells*. Data cells correspond to td[p512] elements, and header cells correspond to th[p513] elements. Cells of both types can have zero or more associated header cells.

It is possible, in certain error cases, for two cells to occupy the same slot.

A **row** is a complete set of slots from $x=0$ to $x=x_{width}-1$, for a particular value of y. Rows usually correspond to tr[p510] elements, though a row group[p516] can have some implied rows[p516] at the end in some cases involving cells[p516] spanning multiple rows.

A **column** is a complete set of slots from $y=0$ to $y=y_{height}-1$, for a particular value of x. Columns can correspond to col[p506] elements. In the absence of col[p506] elements, columns are implied.

A **row group** is a set of rows[p516] anchored at a slot (0, $group_y$) with a particular *height* such that the row group covers all the slots with coordinates (x, y) where $0 \le x < x_{width}$ and $group_y \le y < group_y+height$. Row groups correspond to tbody[p507], thead[p508], and tfoot[p509] elements. Not every row is necessarily in a row group.

A **column group** is a set of columns[p516] anchored at a slot ($group_x$, 0) with a particular *width* such that the column group covers all the slots with coordinates (x, y) where $group_x \le x < group_x+width$ and $0 \le y < y_{height}$. Column groups correspond to colgroup[p505] elements. Not every column is necessarily in a column group.

Row groups[p516] cannot overlap each other. Similarly, column groups[p516] cannot overlap each other.

A cell[p516] cannot cover slots that are from two or more row groups[p516]. It is, however, possible for a cell to be in multiple column groups[p516]. All the slots that form part of one cell are part of zero or one row groups[p516] and zero or more column groups[p516].

In addition to cells[p516], columns[p516], rows[p516], row groups[p516], and column groups[p516], tables[p516] can have a caption[p504] 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[p496] elements and their descendants. Documents must not have table model errors.

4.9.12.1 Forming a table §[p516]

To determine which elements correspond to which slots in a table[p516] associated with a table[p496] element, to determine the dimensions of the table (x_{width} and y_{height}), and to determine if there are any table model errors[p516], user agents must use the following algorithm:

1. Let x_{width} be 0.
2. Let y_{height} be 0.
3. Let *pending* tfoot[p509] *elements* be a list of tfoot[p509] elements, initially empty.
4. Let *the table* be the table[p516] represented by the table[p496] element. The x_{width} and y_{height} variables give *the table*'s dimensions. *The table* is initially empty.
5. If the table[p496] element has no children elements, then return *the table* (which will be empty).
6. Associate the first caption[p504] element child of the table[p496] element with *the table*. If there are no such children, then it has

no associated caption element.

7. Let the *current element* be the first element child of the table element.

   If a step in this algorithm ever requires the *current element* to be **advanced to the next child of the table** when there is no such next child, then the user agent must jump to the step labeled *end*, near the end of this algorithm.

8. While the *current element* is not one of the following elements, advance the *current element* to the next child of the table:

   - colgroup
   - thead
   - tbody
   - tfoot
   - tr

9. If the *current element* is a colgroup, follow these substeps:

   1. *Column groups*: Process the *current element* according to the appropriate case below:

      ↪ **If the *current element* has any col element children**

      Follow these steps:

      1. Let x_{start} have the value of x_{width}.
      2. Let the *current column* be the first col element child of the colgroup element.
      3. *Columns*: If the *current column* col element has a span attribute, then parse its value using the rules for parsing non-negative integers.

         If the result of parsing the value is not an error or zero, then let *span* be that value.

         Otherwise, if the col element has no span attribute, or if trying to parse the attribute's value resulted in an error or zero, then let *span* be 1.

         If *span* is greater than 1000, let it be 1000 instead.
      4. Increase x_{width} by *span*.
      5. Let the last *span* columns in *the table* correspond to the *current column* col element.
      6. If *current column* is not the last col element child of the colgroup element, then let the *current column* be the next col element child of the colgroup element, and return to the step labeled *columns*.
      7. Let all the last columns in *the table* from x=x_{start} to x=x_{width}-1 form a new column group, anchored at the slot (x_{start}, 0), with width x_{width}-x_{start}, corresponding to the colgroup element.

      ↪ **If the *current element* has no col element children**

      1. If the colgroup element has a span attribute, then parse its value using the rules for parsing non-negative integers.

         If the result of parsing the value is not an error or zero, then let *span* be that value.

         Otherwise, if the colgroup element has no span attribute, or if trying to parse the attribute's value resulted in an error or zero, then let *span* be 1.

         If *span* is greater than 1000, let it be 1000 instead.
      2. Increase x_{width} by *span*.
      3. Let the last *span* columns in *the table* form a new column group, anchored at the slot (x_{width}-*span*, 0), with width *span*, corresponding to the colgroup element.

   2. Advance the *current element* to the next child of the table.
   3. While the *current element* is not one of the following elements, advance the *current element* to the next child of the table:

      - colgroup
      - thead

- tbody[p507]
- tfoot[p509]
- tr[p510]

4. If the *current element* is a colgroup[p505] element, jump to the step labeled *column groups* above.

10. Let $y_{current}$ be 0.

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[p517] the *current element* to the next child of the table[p496]:

    - thead[p508]
    - tbody[p507]
    - tfoot[p509]
    - tr[p510]

13. If the *current element* is a tr[p510], then run the algorithm for processing rows[p518], advance[p517] the *current element* to the next child of the table[p496], and return to the step labeled *rows*.

14. Run the algorithm for ending a row group[p518].

15. If the *current element* is a tfoot[p509], then add that element to the list of *pending* tfoot[p509] *elements*, advance[p517] the *current element* to the next child of the table[p496], and return to the step labeled *rows*.

16. The *current element* is either a thead[p508] or a tbody[p507].

    Run the algorithm for processing row groups[p518].

17. Advance[p517] the *current element* to the next child of the table[p496].

18. Return to the step labeled *rows*.

19. *End*: For each tfoot[p509] element in the list of *pending* tfoot[p509] *elements*, in tree order, run the algorithm for processing row groups[p518].

20. If there exists a row[p516] or column[p516] in *the table* containing only slots[p516] that do not have a cell[p516] anchored to them, then this is a table model error[p516].

21. Return *the table*.

The **algorithm for processing row groups**, which is invoked by the set of steps above for processing thead[p508], tbody[p507], and tfoot[p509] elements, is:

1. Let y_{start} have the value of y_{height}.

2. For each tr[p510] element that is a child of the element being processed, in tree order, run the algorithm for processing rows[p518].

3. If $y_{height} > y_{start}$, then let all the last rows[p516] in *the table* from y=y_{start} to y=y_{height}-1 form a new row group[p516], anchored at the slot with coordinate (0, y_{start}), with height y_{height}-y_{start}, corresponding to the element being processed.

4. Run the algorithm for ending a row group[p518].

The **algorithm for ending a row group**, which is invoked by the set of steps above when starting and ending a block of rows, is:

1. While $y_{current}$ is less than y_{height}, follow these steps:

    1. Run the algorithm for growing downward-growing cells[p519].

    2. Increase $y_{current}$ by 1.

2. Empty the *list of downward-growing cells*.

The **algorithm for processing rows**, which is invoked by the set of steps above for processing tr[p510] elements, is:

1. If y_{height} is equal to $y_{current}$, then increase y_{height} by 1. ($y_{current}$ is never *greater* than y_{height}.)

2. Let $x_{current}$ be 0.

3. Run the algorithm for growing downward-growing cells[p519].
4. If the tr[p510] element being processed has no td[p512] or th[p513] element children, then increase $y_{current}$ by 1, abort this set of steps, and return to the algorithm above.
5. Let *current cell* be the first td[p512] or th[p513] element child in the tr[p510] element being processed.
6. *Cells*: While $x_{current}$ is less than x_{width} and the slot with coordinate ($x_{current}$, $y_{current}$) already has a cell assigned to it, increase $x_{current}$ by 1.
7. If $x_{current}$ is equal to x_{width}, increase x_{width} by 1. ($x_{current}$ is never *greater* than x_{width}.)
8. If the *current cell* has a colspan[p515] attribute, then parse that attribute's value[p81], 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 *current cell* has a rowspan[p515] attribute, then parse that attribute's value[p81], 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. Let *cell grows downward* be false.
11. If *rowspan* is zero, then set *cell grows downward* to true and set *rowspan* to 1.
12. If $x_{width} < x_{current}+colspan$, then let x_{width} be $x_{current}+colspan$.
13. If $y_{height} < y_{current}+rowspan$, then let y_{height} be $y_{current}+rowspan$.
14. Let the slots with coordinates (x, y) such that $x_{current} \le x < x_{current}+colspan$ and $y_{current} \le y < y_{current}+rowspan$ be covered by a new cell[p516] c, anchored at ($x_{current}$, $y_{current}$), which has width *colspan* and height *rowspan*, corresponding to the *current cell* element.

    If the *current cell* element is a th[p513] 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[p519] described in the next section.

    If any of the slots involved already had a cell[p516] covering them, then this is a table model error[p516]. Those slots now have two cells overlapping.
15. If *cell grows downward* is true, then add the tuple {c, $x_{current}$, *colspan*} to the *list of downward-growing cells*.
16. Increase $x_{current}$ by *colspan*.
17. If *current cell* is the last td[p512] or th[p513] element child in the tr[p510] element being processed, then increase $y_{current}$ by 1, abort this set of steps, and return to the algorithm above.
18. Let *current cell* be the next td[p512] or th[p513] element child in the tr[p510] element being processed.
19. Return to the step labeled *cells*.

When the algorithms above require the user agent to run the **algorithm for growing downward-growing cells**, the user agent must, for each {*cell*, $cell_x$, *width*} tuple in the *list of downward-growing cells*, if any, extend the cell[p516] *cell* so that it also covers the slots with coordinates (x, $y_{current}$), where $cell_x \le x < cell_x+width$.

4.9.12.2 Forming relationships between data cells and header cells §[p519]

Each cell can be assigned zero or more header cells. The **algorithm for assigning header cells** to a cell *principal cell* is as follows.

1. Let *header list* be an empty list of cells.
2. Let ($principal_x$, $principal_y$) be the coordinate of the slot to which the *principal cell* is anchored.

3. ↪ **If the *principal cell* has a `headers`[p515] attribute specified**

    1. Take the value of the *principal cell*'s `headers`[p515] 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`[p135] with an ID equal to the token is a cell in the same table[p516], and that cell is not the *principal cell*, then add that cell to *header list*.

    ↪ **If *principal cell* does not have a `headers`[p515] 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[p520], with the *principal cell*, the *header list*, the initial coordinate ($principal_x$, y), and the increments $\Delta x=-1$ and $\Delta y=0$.

    4. For each value of x from $principal_x$ to $principal_x+principal_{width}-1$, run the internal algorithm for scanning and assigning header cells[p520], with the *principal cell*, the *header list*, the initial coordinate (x, $principal_y$), and the increments $\Delta x=0$ and $\Delta y=-1$.

    5. If the *principal cell* is anchored in a row group[p516], then add all header cells that are row group headers[p521] 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[p516], then add all header cells that are column group headers[p521] 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[p521] 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 Δ*x* 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[p521], then let *blocked* be true.

   ↪ **If Δ*y* 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[p521], then let *blocked* be true.

5. If *blocked* is false, then add the *current cell* to the *header list*.

↪ **If *current cell* is a data cell and *in header block* is true**
Set *in header block* to false. Add all the cells in *headers from current header block* to the *opaque headers* list, and empty the *headers from current header block* list.

10. Return to the step labeled *loop*.

A header cell anchored at the slot with coordinate (*x*, *y*) with width *width* and height *height* is said to be a **column header** if any of the following are true:

- the cell's scope[p514] attribute is in the Column[p514] state; or
- the cell's scope[p514] attribute is in the Auto[p514] 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 are true:

- the cell's scope[p514] attribute is in the Row[p514] state; or
- the cell's scope[p514] attribute is in the Auto[p514] state, the cell is not a column header[p521], 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[p514] attribute is in the Column Group[p514] state.

A header cell is said to be a **row group header** if its scope[p514] attribute is in the Row Group[p514] 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 §[p521]

This section is non-normative.

The following shows how one might mark up the bottom part of table 45 of the *Smithsonian physical tables, Volume 71*:

```
<table>
 <caption>Specification values: <b>Steel</b>, <b>Castings</b>,
 Ann. A.S.T.M. A27-16, Class B;* P max. 0.06; S max. 0.05.</caption>
 <thead>
  <tr>
   <th rowspan=2>Grade.</th>
   <th rowspan=2>Yield Point.</th>
   <th colspan=2>Ultimate tensile strength</th>
   <th rowspan=2>Per cent elong. 50.8 mm or 2 in.</th>
   <th rowspan=2>Per cent reduct. area.</th>
  </tr>
```

```
  <tr>
   <th>kg/mm<sup>2</sup></th>
   <th>lb/in<sup>2</sup></th>
  </tr>
 </thead>
 <tbody>
  <tr>
   <td>Hard</td>
   <td>0.45 ultimate</td>
   <td>56.2</td>
   <td>80,000</td>
   <td>15</td>
   <td>20</td>
  </tr>
  <tr>
   <td>Medium</td>
   <td>0.45 ultimate</td>
   <td>49.2</td>
   <td>70,000</td>
   <td>18</td>
   <td>25</td>
  </tr>
  <tr>
   <td>Soft</td>
   <td>0.45 ultimate</td>
   <td>42.2</td>
   <td>60,000</td>
   <td>22</td>
   <td>30</td>
  </tr>
 </tbody>
</table>
```

This table could look like this:

Specification values: **Steel, Castings,** Ann. A.S.T.M. A27-16, Class B;* P max. 0.06; S max. 0.05.

Grade.	Yield Point.	Ultimate tensile strength		Per cent elong. 50.8 mm or 2 in.	Per cent reduct. area.
		kg/mm^2	lb/in^2		
Hard.	0.45 ultimate	56.2	80,000	15	20
Medium . . .	0.45 ultimate	49.2	70,000	18	25
Soft	0.45 ultimate	42.2	60,000	22	30

The following shows how one might mark up the gross margin table on page 46 of Apple, Inc's 10-K filing for fiscal year 2008:

```
<table>
 <thead>
  <tr>
   <th>
   <th>2008
   <th>2007
   <th>2006
 <tbody>
  <tr>
   <th>Net sales
   <td>$ 32,479
   <td>$ 24,006
   <td>$ 19,315
  <tr>
   <th>Cost of sales
   <td>  21,334
```

```
   <td>  15,852
   <td>  13,717
 <tbody>
  <tr>
   <th>Gross margin
   <td>$ 11,145
   <td>$  8,154
   <td>$  5,598
 <tfoot>
  <tr>
   <th>Gross margin percentage
   <td>34.3%
   <td>34.0%
   <td>29.0%
</table>
```

This table could look like this:

	2008	2007	2006
Net sales	$ 32,479	$ 24,006	$ 19,315
Cost of sales	21,334	15,852	13,717
Gross margin	$ 11,145	$ 8,154	$ 5,598
Gross margin percentage	34.3%	34.0%	29.0%

The following shows how one might mark up the operating expenses table from lower on the same page of that document:

```
<table>
 <colgroup> <col>
 <colgroup> <col> <col> <col>
 <thead>
  <tr> <th> <th>2008 <th>2007 <th>2006
 <tbody>
  <tr> <th scope=rowgroup> Research and development
       <td> $ 1,109 <td> $ 782 <td> $ 712
  <tr> <th scope=row> Percentage of net sales
       <td> 3.4% <td> 3.3% <td> 3.7%
 <tbody>
  <tr> <th scope=rowgroup> Selling, general, and administrative
       <td> $ 3,761 <td> $ 2,963 <td> $ 2,433
  <tr> <th scope=row> Percentage of net sales
       <td> 11.6% <td> 12.3% <td> 12.6%
</table>
```

This table could look like this:

	2008	2007	2006
Research and development	$ 1,109	$ 782	$ 712
Percentage of net sales	3.4%	3.3%	3.7%
Selling, general, and administrative	$ 3,761	$ 2,963	$ 2,433
Percentage of net sales	11.6%	12.3%	12.6%

4.10 Forms §[p52 3]

4.10.1 Introduction §[p52 3]

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 §[p52 4]

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`[p532] element, inside which are placed the controls. Most controls are represented by the `input`[p539] element, which by default provides a text control. To label a control, the `label`[p536] element is used; the label text and the control itself go inside the `label`[p536] element. Each part of a form is considered a paragraph[p159], and is typically separated from other parts using `p`[p238] elements. Putting this together, here is how one might ask for the customer's name:

```
<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`[p539] element, this time with a `type`[p541] attribute with the value `radio`[p561]. To make the radio buttons work as a group, they are given a common name using the `name`[p628] attribute. To group a batch of controls together, such as, in this case, the radio buttons, one can use the `fieldset`[p620] element. The title of such a group of controls is given by the first element in the `fieldset`[p620], which has to be a `legend`[p623] element.

```
<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`[p539] element with a `type`[p541] attribute with the value `checkbox`[p561]:

```
<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>
 <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>
</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[p539] elements with their type[p541] attribute set to tel[p547]) and email addresses (input[p539] elements with their type[p541] attribute set to email[p548]):

```
<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>
</form>
```

We can use an input[p539] element with its type[p541] attribute set to time[p554] 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[p574], max[p574], and step[p575]. These set the minimum time, the maximum time, and the interval between allowed values (in seconds). This pizzeria only delivers between 11am and 9pm, and doesn't promise anything better than 15 minute increments, which we can mark up as follows:

```
<form>
 <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>
</form>
```

The textarea[p606] 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:

```
<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>
</form>
```

Finally, to make the form submittable we use the button[p585] 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>
 <p><button>Submit order</button></p>
</form>
```

4.10.1.2 Implementing the server-side processing for a form §[p52 6]

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[p632] format, expecting the following parameters sent in an HTTP POST body:

`custname`
: 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 §[p527]

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`[p631] attribute is specified on the `form`[p532] element. This doesn't specify how the form data is encoded, though; to specify that, you use the `enctype`[p632] attribute. You also have to specify the URL of the service that will handle the submitted data, using the `action`[p631] 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`[p628]) also specifies the submission name. Radio buttons can be distinguished from each other in the submission by giving them different values, using the `value`[p543] 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`[p543] attribute.

Given the settings in the previous section, this all becomes:

```
<form method="post"
      enctype="application/x-www-form-urlencoded"
      action="https://pizza.example.com/order.cgi">
 <p><label>Customer name: <input name="custname"></label></p>
 <p><label>Telephone: <input type=tel name="custtel"></label></p>
 <p><label>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 min="11:00" max="21:00" step="900"
name="delivery"></label></p>
 <p><label>Delivery instructions: <textarea name="comments"></textarea></label></p>
 <p><button>Submit order</button></p>
</form>
```

Note

There is no particular significance to the way some of the attributes have their values quoted and others don't. The HTML syntax allows a variety of equally valid ways to specify attributes, as discussed in the syntax section[p1359].

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&de
livery=19%3A00&comments=
```

4.10.1.4 Client-side form validation §[p528]

This section is non-normative.

Forms can be annotated in such a way that the user agent will check the user's input before the form is submitted. The server still has to verify the input is valid (since hostile users can easily bypass the form validation), but it allows the user to avoid the wait incurred by having the server be the sole checker of the user's input.

The simplest annotation is the `required`[p571] attribute, which can be specified on `input`[p539] elements to indicate that the form is not to be submitted until a value is given. By adding this attribute to the customer name, pizza size, and delivery time fields, we allow the user agent to notify the user when the user submits the form without filling in those fields:

```
<form method="post"
      enctype="application/x-www-form-urlencoded"
      action="https://pizza.example.com/order.cgi">
 <p><label>Customer name: <input name="custname" required></label></p>
 <p><label>Telephone: <input type=tel name="custtel"></label></p>
 <p><label>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>
 <p><button>Submit order</button></p>
</form>
```

It is also possible to limit the length of the input, using the `maxlength`[p629] attribute. By adding this to the `textarea`[p606] element, we can limit users to 1000 characters, preventing them from writing huge essays to the busy delivery drivers instead of staying focused and to the point:

```
<form method="post"
      enctype="application/x-www-form-urlencoded"
      action="https://pizza.example.com/order.cgi">
 <p><label>Customer name: <input name="custname" required></label></p>
 <p><label>Telephone: <input type=tel name="custtel"></label></p>
 <p><label>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" maxlength=1000></textarea></label></p>
 <p><button>Submit order</button></p>
</form>
```

Note

When a form is submitted, invalid[p1575] events are fired at each form control that is invalid. This can be useful for displaying a summary of the problems with the form, since typically the browser itself will only report one problem at a time.

4.10.1.5 Enabling client-side automatic filling of form controls §[p529]

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[p633] attribute can be used to describe the field's purpose. In the case of this form, we have three fields that can be usefully annotated in this way: the information about who the pizza is to be delivered to. Adding this information looks like this:

```
<form method="post"
      enctype="application/x-www-form-urlencoded"
      action="https://pizza.example.com/order.cgi">
 <p><label>Customer name: <input name="custname" required autocomplete="shipping name"></label></p>
 <p><label>Telephone: <input type=tel name="custtel" autocomplete="shipping tel"></label></p>
 <p><label>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>
 <p><label>Preferred delivery time: <input type=time min="11:00" max="21:00" step="900"
name="delivery" required></label></p>
 <p><label>Delivery instructions: <textarea name="comments" maxlength=1000></textarea></label></p>
 <p><button>Submit order</button></p>
</form>
```

4.10.1.6 Improving the user experience on mobile devices §[p529]

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[p900] attribute we can select appropriate input modalities:

```
<form method="post"
      enctype="application/x-www-form-urlencoded"
      action="https://pizza.example.com/order.cgi">
```

```
 <p><label>Customer name: <input name="custname" required autocomplete="shipping name"></label></p>
 <p><label>Telephone: <input type=tel name="custtel" autocomplete="shipping tel"></label></p>
 <p><label>Buzzer code: <input name="custbuzz" inputmode="numeric"></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>
 <p><label>Preferred delivery time: <input type=time min="11:00" max="21:00" step="900"
name="delivery" required></label></p>
 <p><label>Delivery instructions: <textarea name="comments" maxlength=1000></textarea></label></p>
 <p><button>Submit order</button></p>
</form>
```

4.10.1.7 The difference between the field type, the autofill field name, and the input modality §[p530]

This section is non-normative.

The [type](p541), [autocomplete](p633), and [inputmode](p900) attributes can seem confusingly similar. For instance, in all three cases, the string "`email`" is a valid value. This section attempts to illustrate the difference between the three attributes and provides advice suggesting how to use them.

The [type](p541) attribute on [input](p539) 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](p539) element, a [textarea](p606) element, a [select](p590) element, etc.

The [autocomplete](p633) 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>](p547). However, which [autocomplete](p633) value to use depends on which phone number the page is asking for, whether they expect a telephone number in the international format or just the local format, and so forth.

For example, a page that forms part of a checkout process on an e-commerce site for a customer buying a gift to be shipped to a friend might need both the buyer's telephone number (in case of payment issues) and the friend's telephone number (in case of delivery issues). If the site expects international phone numbers (with the country code prefix), this could thus look like this:

```
<p><label>Your phone number: <input type=tel name=custtel autocomplete="billing tel"></label>
<p><label>Recipient's phone number: <input type=tel name=shiptel autocomplete="shipping tel"></label>
<p>Please enter complete phone numbers including the country code prefix, as in "+1 555 123 4567".
```

But if the site only supports British customers and recipients, it might instead look like this (notice the use of [tel-national](p638) rather than [tel](p637)):

```
<p><label>Your phone number: <input type=tel name=custtel autocomplete="billing tel-national"></label>
<p><label>Recipient's phone number: <input type=tel name=shiptel autocomplete="shipping tel-
national"></label>
<p>Please enter complete UK phone numbers, as in "(01632) 960 123".
```

Now, consider a person's preferred languages. The right [autocomplete](p633) value is [language](p637). However, there could be a number of different form controls used for the purpose: a text control ([<input type=text>](p546)), a drop-down list ([<select>](p590)), radio buttons

(<input type=radio>[p561]), 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>[p546]. If the page is asking for the user's full name, then the relevant autocomplete[p633] value is name[p636].

```
<p><label>Japanese name: <input name="j" type="text" autocomplete="section-jp name"></label>
<label>Romanized name: <input name="e" type="text" autocomplete="section-en name"></label>
```

In this example, the "section-*[p633]" keywords in the autocomplete[p633] 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.

Separate from the choices regarding type[p541] and autocomplete[p633], the inputmode[p900] 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>[p556], as explained below[p557]; it is instead <input type=text>[p546]. 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 §[p53]1

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[p569] regarding localization of form controls.

4.10.2 Categories §[p53]

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[p157], phrasing content[p157], and interactive content[p158].

A number of the elements are **form-associated elements**, which means they can have a form owner[p626].

⇒ button[p585], fieldset[p620], input[p539], object[p416], output[p611], select[p590], textarea[p606], img[p359], form-associated custom elements[p797]

The form-associated elements[p532] fall into several subcategories:

Listed elements

Denotes elements that are listed in the *form*.elements[p534] and *fieldset*.elements[p622] APIs. These elements also have a form[p627] content attribute, and a matching form[p628] IDL attribute, that allow authors to specify an explicit form owner[p626].

⇒ button[p585], fieldset[p620], input[p539], object[p416], output[p611], select[p590], textarea[p606], form-associated custom elements[p797]

Submittable elements

Denotes elements that can be used for constructing the entry list[p661] when a form[p532] element is submitted[p658].

⇒ button[p585], input[p539], select[p590], textarea[p606], form-associated custom elements[p797]

Some submittable elements[p532] 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[p532] element is reset[p666].

⇒ input[p539], output[p611], select[p590], textarea[p606], form-associated custom elements[p797]

Autocapitalize-and-autocorrect-inheriting elements

Denotes elements that inherit the autocapitalize[p898] and autocorrect[p899] attributes from their form owner[p626].

⇒ button[p585], fieldset[p620], input[p539], output[p611], select[p590], textarea[p606]

Some elements, not all of them form-associated[p532], are categorized as **labelable elements**. These are elements that can be associated with a label[p536] element.

⇒ button[p585], input[p539] (if the type[p541] attribute is *not* in the Hidden[p546] state), meter[p616], output[p611], progress[p614], select[p590], textarea[p606], form-associated custom elements[p797]

4.10.3 The form element §[p53]

Categories[p154]:

Flow content[p157].
Palpable content[p158].

Contexts in which this element can be used[p154]:

Where flow content[p157] is expected.

Content model[p154]:

Flow content[p157], but with no form[p532] element descendants.

Tag omission in text/html[p154]:

Neither tag is omissible.

Content attributes[p154]:

Global attributes[p162]
accept-charset[p533] — Character encodings to use for form submission[p657]
action[p631] — URL to use for form submission[p657]
autocomplete[p533] — Default setting for autofill feature for controls in the form
enctype[p632] — Entry list[p661] encoding type to use for form submission[p657]
method[p631] — Variant to use for form submission[p657]
name[p533] — Name of form to use in the document.forms[p144] API
novalidate[p632] — Bypass form control validation for form submission[p657]
target[p632] — Navigable[p1035] for form submission[p657]
rel[p534]

Accessibility considerations[p154]:

For authors.
For implementers.

Sanitization[p154]:

Uncategorized[p1249] with navigating URL attributes[p1251] action[p631].

DOM interface[p154]:

IDL

```
[Exposed=Window,
 LegacyOverrideBuiltIns,
 LegacyUnenumerableNamedProperties]
interface HTMLFormElement : HTMLElement {
  [HTMLConstructor] constructor();

  [CEReactions, Reflect="accept-charset"] attribute DOMString acceptCharset;
  [CEReactions, ReflectSetter] attribute USVString action;
  [CEReactions] attribute DOMString autocomplete;
  [CEReactions] attribute DOMString enctype;
  [CEReactions] attribute DOMString encoding;
  [CEReactions] attribute DOMString method;
  [CEReactions, Reflect] attribute DOMString name;
  [CEReactions, Reflect] attribute boolean noValidate;
  [CEReactions, Reflect] attribute DOMString target;
  [CEReactions, Reflect] attribute DOMString rel;
  [SameObject, PutForwards=value, Reflect="rel"] 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[p532] element represents[p148] a hyperlink[p313] that can be manipulated through a collection of form-associated elements[p532], 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][p1582]

The **name** attribute represents the form[p532]'s name within the forms[p144] collection. The value must not be the empty string, and the value must be unique amongst the form[p532] elements in the forms[p144] collection that it is in, if any.

The **autocomplete** attribute is an enumerated attribute[p80] with the following keywords and states:

Keyword	State	Brief description
on	On	Form controls will have their autofill field name[p639] set to "on[p635]" by default.
off	Off	Form controls will have their autofill field name[p639] set to "off[p635]" by default.

The attribute's *missing value default*[p80] and *invalid value default*[p80] are both the On[p534] state.

The action[p631], enctype[p632], method[p631], novalidate[p632], and target[p632] attributes are attributes for form submission[p631].

The **rel** attribute on form[p532] elements controls what kinds of links the elements create. The attribute's value must be a unordered set of unique space-separated tokens[p99]. The allowed keywords and their meanings[p326] are defined in an earlier section.

rel[p534]'s supported tokens are the keywords defined in HTML link types[p326] which are allowed on form[p532] elements, impact the processing model, and are supported by the user agent. The possible supported tokens are noreferrer[p338], noopener[p337], and opener[p338]. rel[p534]'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)

***form*.elements**[p534]

Returns an HTMLFormControlsCollection[p117] of the form controls in the form (excluding image buttons for historical reasons).

***form*.length**[p534]

Returns the number of form controls in the form (excluding image buttons for historical reasons).

***form*[*index*]**

Returns the *index*th element in the form (excluding image buttons for historical reasons).

***form*[*name*]**

Returns the form control (or, if there are several, a RadioNodeList[p118] of the form controls) in the form with the given ID or name[p628] (excluding image buttons for historical reasons); or, if there are none, returns the img[p359] 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[p628] changes, for as long as the element remains in the tree.

If there are multiple matching items, then a RadioNodeList[p118] object containing all those elements is returned.

***form*.submit**[p535]**()**

Submits the form, bypassing interactive constraint validation[p653] and without firing a submit[p1576] event.

***form*.requestSubmit**[p536]**([*submitter*])**

Requests to submit the form. Unlike submit()[p535], this method includes interactive constraint validation[p653] and firing a submit[p1576] event, either of which can cancel submission.

The *submitter* argument can be used to point to a specific submit button[p532], whose formaction[p631], formenctype[p632], formmethod[p631], formnovalidate[p632], and formtarget[p632] attributes can impact submission. Additionally, the submitter will be included when constructing the entry list[p661] for submission; normally, buttons are excluded.

***form*.reset**[p536]**()**

Resets the form.

***form*.checkValidity**[p536]**()**

Returns true if the form's controls are all valid; otherwise, returns false.

***form*.reportValidity**[p536]**()**

Returns true if the form's controls are all valid; otherwise, returns false and informs the user.

The **autocomplete** IDL attribute must reflect[p108] the content attribute of the same name, limited to only known values[p109].

The **elements** IDL attribute must return an HTMLFormControlsCollection[p117] rooted at the form[p532] element's root, whose filter matches listed elements[p532] whose form owner[p626] is the form[p532] element, with the exception of input[p539] elements whose type[p541] attribute is in the Image Button[p566] 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[p534] collection.

The supported property indices at any instant are the indices supported by the object returned by the elements[p534] 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 map.

The **submit()** method steps are to submit this from this, with *submitted from submit() method* set to true.

The **`requestSubmit(submitter)`** method, when invoked, must run the following steps:

1. If *submitter* is not null:
    1. If *submitter* is not a submit button[p532], then throw a `TypeError`.
    2. If *submitter*'s form owner[p626] is not this `form`[p532] element, then throw a "`NotFoundError`" `DOMException`.
2. Otherwise, set *submitter* to this `form`[p532] element.
3. Submit[p658] this `form`[p532] element, from *submitter*.

The **`reset()`** method, when invoked, must run the following steps:

1. If the `form`[p532] element is marked as *locked for reset*[p536], then return.
2. Mark the `form`[p532] element as **locked for reset**.
3. Reset[p666] the `form`[p532] element.
4. Unmark the `form`[p532] element as *locked for reset*[p536].

If the **`checkValidity()`** method is invoked, the user agent must statically validate the constraints[p652] of the `form`[p532] element, and return true if the constraint validation returned 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[p653] of the `form`[p532] element, and return true if the constraint validation returned a *positive* result, and false if it returned a *negative* result.

Example

This example shows two search forms:

```
<form action="https://www.google.com/search" method="get">
 <label>Google: <input type="search" name="q"></label> <input type="submit" value="Search...">
</form>
<form action="https://www.bing.com/search" method="get">
 <label>Bing: <input type="search" name="q"></label> <input type="submit" value="Search...">
</form>
```

4.10.4 The `label` element §[p536]

Categories[p154]:

Flow content[p157].
Phrasing content[p157].
Interactive content[p158].
Palpable content[p158].

Contexts in which this element can be used[p154]:

Where phrasing content[p157] is expected.

Content model[p154]:

Phrasing content[p157], but with no descendant labelable elements[p532] unless it is the element's labeled control[p537], and no descendant `label`[p536] elements.

Tag omission in text/html[p154]:

Neither tag is omissible.

Content attributes[p154]:

Global attributes[p162]
`for`[p537] — Associate the label with form control

Accessibility considerations[p154]:

For authors.
For implementers.

Sanitization[p154]**:**

Uncategorized[p1249].

DOM interface[p154]**:**

IDL

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

  readonly attribute HTMLFormElement? form;
  [CEReactions, Reflect="for"] attribute DOMString htmlFor;
  readonly attribute HTMLElement? control;
};
```

The label[p536] element represents[p148] a caption in a user interface. The caption can be associated with a specific form control, known as the label[p536] element's **labeled control**, either using the for[p537] attribute, or by putting the form control inside the label[p536] element itself.

Except where otherwise specified by the following rules, a label[p536] element has no labeled control[p537].

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[p532] in the same tree as the label[p536] element. If the attribute is specified and there is an element in the tree whose ID is equal to the value of the for[p537] attribute, and the first such element in tree order is a labelable element[p532], then that element is the label[p536] element's labeled control[p537].

If the for[p537] attribute is not specified, but the label[p536] element has a labelable element[p532] descendant, then the first such descendant in tree order is the label[p536] element's labeled control[p537].

The label[p536] 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[p536] element for events targeted at interactive content[p158] descendants of a label[p536] element, and any descendants of those interactive content[p158] descendants, must be to do nothing.

Note

Form-associated custom elements[p797] are labelable elements[p532], so for user agents where the label[p536] element's activation behavior impacts the labeled control[p537], 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[p536] in the following snippet could trigger the user agent to fire a click event[p1218] at the input[p539] element, as if the element itself had been triggered by the user:

```
<label><input type=checkbox name=lost> Lost</label>
```

Similarly, assuming `my-checkbox` was declared as a form-associated custom element[p797] (like in this example[p786]), then the code

```
<label><my-checkbox name=lost></my-checkbox> Lost</label>
```

would have the same behavior, firing a click event[p1218] 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.

```
<p><label>Full name: <input name=fn> <small>Format: First Last</small></label></p>
<p><label>Age: <input name=age type=number min=0></label></p>
<p><label>Post code: <input name=pc> <small>Format: AB12 3CD</small></label></p>
```

For web developers (non-normative)

***label*.control**[p538]

Returns the form control that is associated with this element.

***label*.form**[p538]

Returns the form owner[p626] of the form control that is associated with this element.

Returns null if there isn't one.

The **control** IDL attribute must return the label[p536] element's labeled control[p537], if any, or null if there isn't one.

The **form** IDL attribute must run the following steps:

1. If the label[p536] element has no labeled control[p537], then return null.
2. If the label[p536] element's labeled control[p537] is not a form-associated element[p532], then return null.
3. Return the label[p536] element's labeled control[p537]'s form owner[p626] (which can still be null).

Note

The form[p538] IDL attribute on the label[p536] element is different from the form[p627] IDL attribute on listed[p532] form-associated elements[p532], and the label[p536] element does not have a form[p627] content attribute.

For web developers (non-normative)

***control*.labels**[p538]

Returns a NodeList of all the label[p536] elements that the form control is associated with.

Labelable elements[p532] and all input[p539] elements have a live[p48] NodeList object associated with them that represents the list of label[p536] elements, in tree order, whose labeled control[p537] is the element in question. The **labels** IDL attribute of labelable elements[p532] that are not form-associated custom elements[p797], and the labels[p538] IDL attribute of input[p539] elements, on getting, must return that NodeList object, and that same value must always be returned, unless this element is an input[p539] element whose type[p541] attribute is in the Hidden[p546] state, in which case it must instead return null.

Form-associated custom elements[p797] don't have a labels[p538] IDL attribute. Instead, their ElementInternals[p809] object has a **labels** IDL attribute. On getting, it must throw a "NotSupportedError" DOMException if the target element[p810] is not a form-associated custom element[p797]. 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[p538] returns when an input[p539] element has its type[p541] attribute changed.

```
<!doctype html>
<p><label><input></label></p>
<script>
 const input = document.querySelector('input');
 const labels = input.labels;
 console.assert(labels.length === 1);

 input.type = 'hidden';
 console.assert(labels.length === 0); // the input is no longer the label's labeled control
 console.assert(input.labels === null);

 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 §[p539]

Categories[p154]:

Flow content[p157].
Phrasing content[p157].
If the `type`[p541] attribute is *not* in the Hidden[p546] state: Interactive content[p158].
If the `type`[p541] attribute is *not* in the Hidden[p546] state: Listed[p532], labelable[p532], submittable[p532], resettable[p532], and autocapitalize-and-autocorrect inheriting[p532] form-associated element[p532].
If the `type`[p541] attribute is in the Hidden[p546] state: Listed[p532], submittable[p532], resettable[p532], and autocapitalize-and-autocorrect inheriting[p532] form-associated element[p532].
If the `type`[p541] attribute is *not* in the Hidden[p546] state: Palpable content[p158].

Contexts in which this element can be used[p154]:

Where phrasing content[p157] is expected.

Content model[p154]:

Nothing[p155].

Tag omission in text/html[p154]:

No end tag[p1359].

Content attributes[p154]:

Global attributes[p162]
`accept`[p563] — Hint for expected file type in file upload controls[p563]
`alpha`[p559] — Allow the color's alpha component to be set
`alt`[p567] — Replacement text for use when images are not available
`autocomplete`[p633] — Hint for form autofill feature
`checked`[p543] — Whether the control is checked
`colorspace`[p560] — The color space of the serialized color
`dirname`[p629] — Name of form control to use for sending the element's directionality[p168] in form submission[p657]
`disabled`[p630] — Whether the form control is disabled
`form`[p627] — Associates the element with a `form`[p532] element
`formaction`[p631] — URL to use for form submission[p657]
`formenctype`[p632] — Entry list[p661] encoding type to use for form submission[p657]
`formmethod`[p631] — Variant to use for form submission[p657]
`formnovalidate`[p632] — Bypass form control validation for form submission[p657]
`formtarget`[p632] — Navigable[p1035] for form submission[p657]
`height`[p495] — Vertical dimension
`list`[p576] — List of autocomplete options
`max`[p574] — Maximum value
`maxlength`[p569] — Maximum length of value
`min`[p574] — Minimum value
`minlength`[p569] — Minimum length of value
`multiple`[p572] — Whether to allow multiple values
`name`[p628] — Name of the element to use for form submission[p657] and in the `form.elements`[p534] API
`pattern`[p573] — Pattern to be matched by the form control's value
`placeholder`[p578] — User-visible label to be placed within the form control
`popovertarget`[p935] — Targets a popover element to toggle, show, or hide
`popovertargetaction`[p935] — Indicates whether a targeted popover element is to be toggled, shown, or hidden
`readonly`[p570] — Whether to allow the value to be edited by the user
`required`[p571] — Whether the control is required for form submission[p657]
`size`[p570] — Size of the control
`src`[p566] — Address of the resource
`step`[p575] — Granularity to be matched by the form control's value
`type`[p541] — Type of form control
`value`[p543] — Value of the form control
`width`[p495] — Horizontal dimension
Also, the `title`[p573] attribute has special semantics[p573] on this element: Description of pattern (when used with `pattern`[p573] attribute)

Accessibility considerations[p154]:

`type`[p541] attribute in the Hidden[p546] state: for authors; for implementers.
`type`[p541] attribute in the Text[p546] state: for authors; for implementers.

type[p541] attribute in the Search[p546] state: for authors; for implementers.
type[p541] attribute in the Telephone[p547] state: for authors; for implementers.
type[p541] attribute in the URL[p547] state: for authors; for implementers.
type[p541] attribute in the Email[p548] state: for authors; for implementers.
type[p541] attribute in the Password[p550] state: for authors; for implementers.
type[p541] attribute in the Date[p550] state: for authors; for implementers.
type[p541] attribute in the Month[p552] state: for authors; for implementers.
type[p541] attribute in the Week[p553] state: for authors; for implementers.
type[p541] attribute in the Time[p554] state: for authors; for implementers.
type[p541] attribute in the Local Date and Time[p554] state: for authors; for implementers.
type[p541] attribute in the Number[p556] state: for authors; for implementers.
type[p541] attribute in the Range[p557] state: for authors; for implementers.
type[p541] attribute in the Color[p559] state: for authors; for implementers.
type[p541] attribute in the Checkbox[p561] state: for authors; for implementers.
type[p541] attribute in the Radio Button[p561] state: for authors; for implementers.
type[p541] attribute in the File Upload[p563] state: for authors; for implementers.
type[p541] attribute in the Submit Button[p565] state: for authors; for implementers.
type[p541] attribute in the Image Button[p566] state: for authors; for implementers.
type[p541] attribute in the Reset Button[p568] state: for authors; for implementers.
type[p541] attribute in the Button[p569] state: for authors; for implementers.

Sanitization[p154]:

Uncategorized[p1249] with navigating URL attributes[p1251] formaction[p631].

DOM interface[p154]:

IDL

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

  [CEReactions, Reflect] attribute DOMString accept;
  [CEReactions, Reflect] attribute boolean alpha;
  [CEReactions, Reflect] attribute DOMString alt;
  [CEReactions, ReflectSetter] attribute DOMString autocomplete;
  [CEReactions, Reflect="checked"] attribute boolean defaultChecked;
  attribute boolean checked;
  [CEReactions] attribute DOMString colorSpace;
  [CEReactions, Reflect] attribute DOMString dirName;
  [CEReactions, Reflect] attribute boolean disabled;
  readonly attribute HTMLFormElement? form;
  attribute FileList? files;
  [CEReactions, ReflectSetter] attribute USVString formAction;
  [CEReactions] attribute DOMString formEnctype;
  [CEReactions] attribute DOMString formMethod;
  [CEReactions, Reflect] attribute boolean formNoValidate;
  [CEReactions, Reflect] attribute DOMString formTarget;
  [CEReactions, ReflectSetter] attribute unsigned long height;
  attribute boolean indeterminate;
  readonly attribute HTMLDataListElement? list;
  [CEReactions, Reflect] attribute DOMString max;
  [CEReactions, ReflectNonNegative] attribute long maxLength;
  [CEReactions, Reflect] attribute DOMString min;
  [CEReactions, ReflectNonNegative] attribute long minLength;
  [CEReactions, Reflect] attribute boolean multiple;
  [CEReactions, Reflect] attribute DOMString name;
  [CEReactions, Reflect] attribute DOMString pattern;
  [CEReactions, Reflect] attribute DOMString placeholder;
  [CEReactions, Reflect] attribute boolean readOnly;
  [CEReactions, Reflect] attribute boolean required;
  [CEReactions, Reflect] attribute unsigned long size;
  [CEReactions, ReflectURL] attribute USVString src;
  [CEReactions, Reflect] attribute DOMString step;
```

```
  [CEReactions] attribute DOMString type;
  [CEReactions, Reflect="value"] attribute DOMString defaultValue;
  [CEReactions] attribute [LegacyNullToEmptyString] DOMString value;
  attribute object? valueAsDate;
  attribute unrestricted double valueAsNumber;
  [CEReactions, ReflectSetter] attribute unsigned long width;

  undefined stepUp(optional long n = 1);
  undefined stepDown(optional long n = 1);

  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);

  undefined showPicker();

  // also has obsolete members
};
HTMLInputElement includes PopoverTargetAttributes;
```

The input[p539] element represents[p148] 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[p80] with the following keywords and states:

Keyword	State	Data type	Control type
hidden	Hidden[p546]	An arbitrary string	n/a
text	Text[p546]	Text with no line breaks	A text control
search	Search[p546]	Text with no line breaks	Search control
tel	Telephone[p547]	Text with no line breaks	A text control
url	URL[p547]	An absolute URL	A text control
email	Email[p548]	An email address or list of email addresses	A text control
password	Password[p550]	Text with no line breaks (sensitive information)	A text control that obscures data entry
date	Date[p550]	A date (year, month, day) with no time zone	A date control
month	Month[p552]	A date consisting of a year and a month with no time zone	A month control
week	Week[p553]	A date consisting of a week-year number and a week number with no time zone	A week control
time	Time[p554]	A time (hour, minute, seconds, fractional seconds) with no time zone	A time control
datetime-local	Local Date and Time[p554]	A date and time (year, month, day, hour, minute, second, fraction of a second) with no time zone	A date and time control
number	Number[p556]	A numerical value	A text control or spinner control
range	Range[p557]	A numerical value, with the extra semantic that the exact value is not important	A slider control or similar
color	Color[p559]	An sRGB color with 8-bit red, green, and blue components	A color picker

Keyword	State	Data type	Control type
checkbox	Checkbox[p561]	A set of zero or more values from a predefined list	A checkbox
radio	Radio Button[p561]	An enumerated value	A radio button
file	File Upload[p563]	Zero or more files each with a MIME type and optionally a filename	A label and a button
submit	Submit Button[p565]	An enumerated value, with the extra semantic that it must be the last value selected and initiates form submission	A button
image	Image Button[p566]	A coordinate, relative to a particular image's size, with the extra semantic that it must be the last value selected and initiates form submission	Either a clickable image, or a button
reset	Reset Button[p568]	n/a	A button
button	Button[p569]	n/a	A button

The attribute's *missing value default*[p80] and *invalid value default*[p80] are both the Text[p546] state.

Which of the accept[p563], alpha[p559], alt[p567], autocomplete[p633], checked[p543], colorspace[p560], dirname[p629], formaction[p631], formenctype[p632], formmethod[p631], formnovalidate[p632], formtarget[p632], height[p495], list[p576], max[p574], maxlength[p569], min[p574], minlength[p569], multiple[p572], pattern[p573], placeholder[p578], readonly[p570], required[p571], size[p570], src[p566], step[p575], and width[p495] content attributes, the checked[p581], files[p581], valueAsDate[p581], valueAsNumber[p581], and list[p582] IDL attributes, the select()[p648] method, the selectionStart[p648], selectionEnd[p649], and selectionDirection[p649], IDL attributes, the setRangeText()[p650] and setSelectionRange()[p649] methods, the stepUp()[p581] and stepDown()[p581] methods, and the input and change[p1575] events **apply** to an input[p539] element depends on the state of its type[p541] 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[p569], for APIs[p579], for events[p583]).

The following table is non-normative and summarizes which of those content attributes, IDL attributes, methods, and events apply[p542] to each state:

	Hidden[p546]	Text[p546], Search[p546]	Telephone[p547], URL[p547]	Email[p548]	Password[p550]	Date[p550], Month[p552], Week[p553], Time[p554]	Local Date and Time[p554]	Number[p556]	Range[p557]	Color[p559]	Checkbox[p561], Radio Button[p561]
Content attributes											
accept[p563]	·	·	·	·	·	·	·	·	·	·	·
alpha[p559]	·	·	·	·	·	·	·	·	·	Yes	·
alt[p567]	·	·	·	·	·	·	·	·	·	·	·
autocomplete[p633]	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	·
checked[p543]	·	·	·	·	·	·	·	·	·	·	Yes
colorspace[p560]	·	·	·	·	·	·	·	·	·	Yes	·
dirname[p629]	Yes	Yes	Yes	Yes	Yes	·	·	·	·	·	·
formaction[p631]	·	·	·	·	·	·	·	·	·	·	·
formenctype[p632]	·	·	·	·	·	·	·	·	·	·	·
formmethod[p631]	·	·	·	·	·	·	·	·	·	·	·
formnovalidate[p632]	·	·	·	·	·	·	·	·	·	·	·
formtarget[p632]	·	·	·	·	·	·	·	·	·	·	·
height[p495]	·	·	·	·	·	·	·	·	·	·	·
list[p576]	·	Yes	Yes	Yes	·	Yes	Yes	Yes	Yes	Yes	·
max[p574]	·	·	·	·	·	Yes	Yes	Yes	Yes	·	·
maxlength[p569]	·	Yes	Yes	Yes	Yes	·	·	·	·	·	·
min[p574]	·	·	·	·	·	Yes	Yes	Yes	Yes	·	·
minlength[p569]	·	Yes	Yes	Yes	Yes	·	·	·	·	·	·
multiple[p572]	·	·	·	Yes	·	·	·	·	·	·	·
pattern[p573]	·	Yes	Yes	Yes	Yes	·	·	·	·	·	·
placeholder[p578]	·	Yes	Yes	Yes	Yes	·	·	Yes	·	·	·
popovertarget[p935]	·	·	·	·	·	·	·	·	·	·	·
popovertargetaction[p935]	·	·	·	·	·	·	·	·	·	·	·
readonly[p570]	·	Yes	Yes	Yes	Yes	Yes	Yes	Yes	·	·	·
required[p571]	·	Yes	Yes	Yes	Yes	Yes	Yes	Yes	·	·	Yes
size[p570]	·	Yes	Yes	Yes	Yes	·	·	·	·	·	·
src[p566]	·	·	·	·	·	·	·	·	·	·	·
step[p575]	·	·	·	·	·	Yes	Yes	Yes	Yes	·	·

	Hidden	Text, Search	Telephone, URL	Email	Password	Date, Month, Week, Time	Local Date and Time	Number	Range	Color	Checkbox, Radio Button
width	·	·	·	·	·	·	·	·	·	·	·
IDL attributes and methods											
checked	·	·	·	·	·	·	·	·	·	·	Yes
files	·	·	·	·	·	·	·	·	·	·	·
value	default	value	value	value	value	value	value	value	value	value	default/on
valueAsDate	·	·	·	·	·	Yes	·	·	·	·	·
valueAsNumber	·	·	·	·	·	Yes	Yes	Yes	Yes	·	·
list	·	Yes	Yes	Yes	·	Yes	Yes	Yes	Yes	Yes	·
select()	·	Yes	Yes	Yes†	Yes	Yes†	Yes†	Yes†	·	Yes†	·
selectionStart	·	Yes	Yes	·	Yes	·	·	·	·	·	·
selectionEnd	·	Yes	Yes	·	Yes	·	·	·	·	·	·
selectionDirection	·	Yes	Yes	·	Yes	·	·	·	·	·	·
setRangeText()	·	Yes	Yes	·	Yes	·	·	·	·	·	·
setSelectionRange()	·	Yes	Yes	·	Yes	·	·	·	·	·	·
stepDown()	·	·	·	·	·	Yes	Yes	Yes	Yes	·	·
stepUp()	·	·	·	·	·	Yes	Yes	Yes	Yes	·	·
Events											
input event	·	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes
change event	·	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes

† If the control has no selectable text, the select() method results in a no-op, with no "InvalidStateError" DOMException.

Some states of the type attribute define a **value sanitization algorithm**.

Each input element has a value, which is exposed by the value IDL attribute. Some states define an **algorithm to convert a string to a number**, an **algorithm to convert a number to a string**, an **algorithm to convert a string to a Date object**, and an **algorithm to convert a Date object to a string**, which are used by max, min, step, valueAsDate, valueAsNumber, 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 its user validity, dirty value flag, and dirty checkedness flag back to false, set the value of the element to the value of the value content attribute, if there is one, or the empty string otherwise, set the checkedness of the element to true if the element has a checked content attribute and false if it does not, empty the list of selected files, and then invoke the value sanitization algorithm, if the type attribute's current state defines one.

Each input element can be mutable. Except where otherwise specified, an input element is always mutable. Similarly, except where otherwise specified, the user agent should not allow the user to modify the element's value or checkedness.

When an input element is disabled, it is not mutable.

Note

The readonly attribute can also in some cases (e.g. for the Date state, but not the Checkbox state) stop an input element from being mutable.

The input element can **support a picker**. A picker is a user interface element that allows the end user to choose a value. Whether an input element supports a picker depends on the type attribute state and implementation-defined behavior. An input element must support a picker when its type attribute is in the File Upload state.

Note

As of the time of this writing, typical browser implementations show such picker UI for:

- *input elements whose type attributes are in the Date, Month, Week, Time, Local Date and Time, and Color states;*
- *input elements in various states that have a suggestions source element;*
- *input elements whose type attribute is in the File Upload state; and*
- *select elements.*

The cloning steps for input elements given *node*, *copy*, and *subtree* are to propagate the value, dirty value flag, checkedness, dirty checkedness flag, and indeterminateness from *node* to *copy*.

The activation behavior for input elements *element*, given *event*, are these steps:

1. If *element* is not mutable, and *element*'s type attribute is neither in the Checkbox nor in the Radio state, then return.
2. Run *element*'s **input activation behavior**, if any, and do nothing otherwise.
3. If *element* has a form owner and *element*'s type attribute is not in the Button state, then return.
4. Run the popover target attribute activation behavior given *element* and *event*'s target.

Note

Recall that an element's activation behavior runs for both user-initiated activations and for synthetic activations (e.g., via `el.click()`). User agents might also have behaviors for a given control — not specified here — that are triggered only by true user-initiated activations. A common choice is to show the picker, if applicable, for the control. In contrast, the input activation behavior only shows pickers for the special historical cases of the File Upload and Color states.

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 indeterminateness 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-canceled-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 indeterminateness back to the values they had before the legacy-pre-activation behavior was run.
2. If this element's type attribute is in the Radio Button state, then if the element to which a reference was obtained in the legacy-pre-activation behavior, if any, is still in what is now this element's radio button group, if it still has one, and if so, set that element's checkedness to true; or else, if there was no such element, or that element is no longer in this element's radio button group, or if this element no longer has a radio button group, set this element's checkedness to false.

When an input element is first created, the element's rendering and behavior must be set to the rendering and behavior defined for the type attribute's state, and the value sanitization algorithm, if one is defined for the type attribute's state, must be invoked.

When an input element's type attribute changes state, the user agent must run the following steps:

1. If the previous state of the element's type attribute put the value IDL attribute in the *value* mode, and the element's value is not the empty string, and the new state of the element's type attribute puts the value IDL attribute in either the *default* mode or the *default/on* mode, then set the element's value content attribute to the element's value.

2. Otherwise, if the previous state of the element's type attribute put the value IDL attribute in any mode other than the *value* mode, and the new state of the element's type attribute puts the value IDL attribute in the *value* mode, then set the value of the element to the value of the value content attribute, if there is one, or the empty string otherwise, and then set the control's dirty value flag to false.

3. Otherwise, if the previous state of the element's type attribute put the value IDL attribute in any mode other than the *filename* mode, and the new state of the element's type attribute puts the value IDL attribute in the *filename* mode, then set the value of the element to the empty string.

4. Update the element's rendering and behavior to the new state's.

5. **Signal a type change** for the element. (The Radio Button state uses this, in particular.)

6. Invoke the value sanitization algorithm, if one is defined for the type attribute's new state.

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 dirname attribute controls how the element's directionality is submitted. The disabled attribute is used to make the control non-interactive and to prevent its value from being submitted. The form attribute is used to explicitly associate the input element with its form owner. The autocomplete attribute controls how the user agent provides autofill behavior.

Each input element has an **indeterminateness**, a boolean which is initially false. It only has an effect on checkbox controls, including their rendering and accessibility semantics. An input element's indeterminateness is independent of its checkedness.

The **indeterminate** getter steps are to return this's indeterminateness.

The indeterminate setter steps are to set this's indeterminateness to the given value.

The **colorSpace** IDL attribute must reflect the colorspace content attribute, limited to only known values. The **type** IDL attribute must reflect the respective content attribute of the same name, limited to only known values.

The **width** and **height** getter steps are to return the respective component of the result of determining the dimensions of this.

To determine the **dimensions** of an input element *input*:

1. If *input*'s type attribute is not in the Image Button state, then return a width of 0 and a height of 0. (In that case no image is *available*.)

2. Let *naturalDimensions* be null.

3. If an image is *available*, then set *naturalDimensions* to the result of applying the default sizing algorithm with that image's natural width, natural height, and natural aspect ratio, if any, using a default object size of 300 by 150.

4. Return the result of determining the dimensions of *input* given *naturalDimensions*.

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 elements. 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 and dirname content attributes apply 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, alpha, alt, checked, colorspace, formaction, formenctype, formmethod, formnovalidate, formtarget, height, list, max, maxlength, min, minlength, multiple, pattern, placeholder, popovertarget, popovertargetaction, 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 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+000A LINE FEED (LF) or U+000D CARRIAGE RETURN (CR) characters into the element's value.

If the element is mutable, the user agent should allow the user to change the writing direction of the element, setting it either to a left-to-right writing direction or a right-to-left writing direction. If the user does so, the user agent must then run the following steps:

1. Set the element's dir attribute to "ltr" if the user selected a left-to-right writing direction, and "rtl" if the user selected a right-to-left writing direction.
2. Queue 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 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, dirname, 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[p542] to the element: accept[p563], alpha[p559], alt[p567], checked[p543], colorspace[p560], formaction[p631], formenctype[p632], formmethod[p631], formnovalidate[p632], formtarget[p632], height[p495], max[p574], min[p574], multiple[p572], popovertarget[p935], popovertargetaction[p935], src[p566], step[p575], and width[p495].
- The following IDL attributes and methods do not apply[p542] to the element: checked[p581], files[p581], valueAsDate[p581], and valueAsNumber[p581] IDL attributes; stepDown()[p581] and stepUp()[p581] methods.

4.10.5.1.3 Telephone state (type=tel) §[p54]

When an input[p539] element's type[p541] attribute is in the Telephone[p547] state, the rules in this section apply.

The input[p539] element represents[p148] a control for editing a telephone number given in the element's value[p626].

If the element is mutable[p626], its value[p626] should be editable by the user. User agents may change the spacing and, with care, the punctuation of values[p626] that the user enters. User agents must not allow users to insert U+000A LINE FEED (LF) or U+000D CARRIAGE RETURN (CR) characters into the element's value[p626].

The value[p543] 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[p543] is as follows: Strip newlines from the value[p626].

Note

Unlike the URL[p547] and Email[p548] types, the Telephone[p547] 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[p573] attribute or the setCustomValidity()[p654] method to hook into the client-side validation mechanism.

Bookkeeping details

- The following common input[p539] element content attributes, IDL attributes, and methods apply[p542] to the element: autocomplete[p633], dirname[p629], list[p576], maxlength[p569], minlength[p569], pattern[p573], placeholder[p578], readonly[p570], required[p571], and size[p570] content attributes; list[p582], selectionStart[p648], selectionEnd[p649], selectionDirection[p649], and value[p580] IDL attributes; select()[p648], setRangeText()[p650], and setSelectionRange()[p649] methods.
- The value[p580] IDL attribute is in mode value[p580].
- The input and change[p1575] events apply[p542].
- The following content attributes must not be specified and do not apply[p542] to the element: accept[p563], alpha[p559], alt[p567], checked[p543], colorspace[p560], formaction[p631], formenctype[p632], formmethod[p631], formnovalidate[p632], formtarget[p632], height[p495], max[p574], min[p574], multiple[p572], popovertarget[p935], popovertargetaction[p935], src[p566], step[p575], and width[p495].
- The following IDL attributes and methods do not apply[p542] to the element: checked[p581], files[p581], valueAsDate[p581], and valueAsNumber[p581] IDL attributes; stepDown()[p581] and stepUp()[p581] methods.

4.10.5.1.4 URL state (type=url) §[p54]

When an input[p539] element's type[p541] attribute is in the URL[p547] state, the rules in this section apply.

The input[p539] element represents[p148] a control for editing a single absolute URL given in the element's value[p626].

If the element is mutable[p626], the user agent should allow the user to change the URL represented by its value[p626]. User agents may allow the user to set the value[p626] 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[p626] 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[p626] 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[p626].

The value[p543] attribute, if specified and not empty, must have a value that is a valid URL potentially surrounded by spaces[p100] that is also an absolute URL.

The value sanitization algorithm[p543] is as follows: Strip newlines from the value[p626], then strip leading and trailing ASCII whitespace from the value[p626].

Constraint validation: While the value[p626] of the element is neither the empty string nor a valid absolute URL, the element is suffering from a type mismatch[p651].

Bookkeeping details

- The following common input element content attributes, IDL attributes, and methods apply to the element: autocomplete, dirname, 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, alpha, alt, checked, colorspace, formaction, formenctype, formmethod, formnovalidate, formtarget, height, max, min, multiple, popovertarget, popovertargetaction, 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:

```
<input type="url" name="location" list="urls">
<datalist id="urls">
 <option label="MIME: Format of Internet Message Bodies" value="https://www.rfc-editor.org/rfc/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://mediasession.spec.whatwg.org/">
 <option label="The Single UNIX Specification, Version 3" value="http://www.unix.org/version3/">
</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, sorted in some implementation-defined manner (maybe by how frequently the user refers to those URLs). Note how the UA is using the knowledge that the values are URLs to allow the user to omit the scheme part and perform intelligent matching on the domain name.

The last two URLs (and probably many more, given the scrollbar's indications of more values being available) are the matches from the user agent's session history data. This data is not made available to the page DOM. In this particular case, the UA has no titles to provide for those values.

4.10.5.1.5 Email state (`type=email`)

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[p626] 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[p626]. User agents may transform the value[p626] for display and editing; in particular, user agents should convert punycode in the domain labels of the value[p626] 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[p652].

The `value`[p543] attribute, if specified and not empty, must have a value that is a single valid email address[p549].

The value sanitization algorithm[p543] is as follows: Strip newlines from the value[p626], then strip leading and trailing ASCII whitespace from the value[p626].

Constraint validation: While the value[p626] of the element is neither the empty string nor a single valid email address[p549], the element is suffering from a type mismatch[p651].

↪ **When the `multiple`[p572] attribute *is* specified on the element**

The `input`[p539] element represents[p148] a control for adding, removing, and editing the email addresses given in the element's values[p626].

If the element is mutable[p626], the user agent should allow the user to add, remove, and edit the email addresses represented by its values[p626]. User agents may allow the user to set any individual value in the list of values[p626] to a string that is not a valid email address[p549], 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[p626]. User agents may transform the values[p626] for display and editing; in particular, user agents should convert punycode in the domain labels of the value[p626] 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[p652].

Whenever the user changes the element's values[p626], the user agent must run the following steps:

1. Let *latest values* be a copy of the element's values[p626].
2. Strip leading and trailing ASCII whitespace from each value in *latest values*.
3. Set the element's value[p626] to 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`[p543] attribute, if specified, must have a value that is a valid email address list[p550].

The value sanitization algorithm[p543] is as follows:

1. Split on commas the element's value[p626], strip leading and trailing ASCII whitespace from each resulting token, if any, and let the element's values[p626] be the (possibly empty) resulting list of (possibly empty) tokens, maintaining the original order.
2. Set the element's value[p626] to the result of concatenating the element's values[p626], separating each value from the next by a single U+002C COMMA character (,), maintaining the list's order.

Constraint validation: While the value[p626] of the element is not a valid email address list[p550], the element is suffering from a type mismatch[p651].

When the `multiple`[p572] attribute is set or removed, the user agent must run the value sanitization algorithm[p543].

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][p1579] [RFC5322][p1586] [RFC1034][p1585] [RFC1123][p1585]

```
email         = 1*( atext / "." ) "@" label *( "." label )
label         = let-dig [ [ ldh-str ] let-dig ]  ; limited to a length of 63 characters by RFC 1034 section 3.5
atext         = < as defined in RFC 5322 section 3.2.3 >
let-dig       = < as defined in RFC 1034 section 3.5 >
ldh-str       = < as defined in RFC 1034 section 3.5 >
```

Note

This requirement is a willful violation of RFC 5322, which defines a syntax for 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.

```
/^[a-zA-Z0-9.!#$%&'*+\/=?^_`{|}~-]+@[a-zA-Z0-9](?:[a-zA-Z0-9-]{0,61}[a-zA-Z0-9])?(?:\.[a-zA-Z0-9](?:[a-zA-Z0-9-]{0,61}[a-zA-Z0-9])?)*$/
```

A **valid email address list** is a set of comma-separated tokens[p99], where each token is itself a valid email address[p549]. To obtain the list of tokens from a valid email address list[p550], an implementation must split the string on commas.

Bookkeeping details

- The following common `input`[p539] element content attributes, IDL attributes, and methods apply[p542] to the element: `autocomplete`[p633], `dirname`[p629], `list`[p576], `maxlength`[p569], `minlength`[p569], `multiple`[p572], `pattern`[p573], `placeholder`[p578], `readonly`[p570], `required`[p571], and `size`[p570] content attributes; `list`[p582] and `value`[p580] IDL attributes; `select()`[p648] method.
- The `value`[p580] IDL attribute is in mode value[p580].
- The `input` and `change`[p1575] events apply[p542].
- The following content attributes must not be specified and do not apply[p542] to the element: `accept`[p563], `alpha`[p559], `alt`[p567], `checked`[p543], `colorspace`[p560], `formaction`[p631], `formenctype`[p632], `formmethod`[p631], `formnovalidate`[p632], `formtarget`[p632], `height`[p495], `max`[p574], `min`[p574], `popovertarget`[p935], `popovertargetaction`[p935], `src`[p566], `step`[p575], and `width`[p495].
- The following IDL attributes and methods do not apply[p542] to the element: `checked`[p581], `files`[p581], `selectionStart`[p648], `selectionEnd`[p649], `selectionDirection`[p649], `valueAsDate`[p581], and `valueAsNumber`[p581] IDL attributes; `setRangeText()`[p650], `setSelectionRange()`[p649], `stepDown()`[p581] and `stepUp()`[p581] methods.

4.10.5.1.6 Password state (`type=password`) §[p55]

When an `input`[p539] element's `type`[p541] attribute is in the Password[p550] state, the rules in this section apply.

The `input`[p539] element represents[p148] a one line plain text edit control for the element's value[p626]. The user agent should obscure the value so that people other than the user cannot see it.

If the element is mutable[p626], its value[p626] 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[p626].

The `value`[p543] 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[p543] is as follows: Strip newlines from the value[p626].

Bookkeeping details

- The following common `input`[p539] element content attributes, IDL attributes, and methods apply[p542] to the element: `autocomplete`[p633], `dirname`[p629], `maxlength`[p569], `minlength`[p569], `pattern`[p573], `placeholder`[p578], `readonly`[p570], `required`[p571], and `size`[p570] content attributes; `selectionStart`[p648], `selectionEnd`[p649], `selectionDirection`[p649], and `value`[p580] IDL attributes; `select()`[p648], `setRangeText()`[p650], and `setSelectionRange()`[p649] methods.
- The `value`[p580] IDL attribute is in mode value[p580].
- The `input` and `change`[p1575] events apply[p542].
- The following content attributes must not be specified and do not apply[p542] to the element: `accept`[p563], `alpha`[p559], `alt`[p567], `checked`[p543], `colorspace`[p560], `formaction`[p631], `formenctype`[p632], `formmethod`[p631], `formnovalidate`[p632], `formtarget`[p632], `height`[p495], `list`[p576], `max`[p574], `min`[p574], `multiple`[p572], `popovertarget`[p935], `popovertargetaction`[p935], `src`[p566], `step`[p575], and `width`[p495].
- The following IDL attributes and methods do not apply[p542] to the element: `checked`[p581], `files`[p581], `list`[p582], `valueAsDate`[p581], and `valueAsNumber`[p581] IDL attributes; `stepDown()`[p581] and `stepUp()`[p581] methods.

4.10.5.1.7 Date state (`type=date`) §[p55]

When an `input`[p539] element's `type`[p541] attribute is in the Date[p550] state, the rules in this section apply.

The `input`[p539] element represents[p148] a control for setting the element's value[p626] to a string representing a specific date[p87].

If the element is mutable[p626], the user agent should allow the user to change the date[p87] represented by its value[p626], as obtained by parsing a date[p87] from it. User agents must not allow the user to set the value[p626] to a non-empty string that is not a valid date string[p87]. If the user agent provides a user interface for selecting a date[p87], then the value[p626] must be set to a valid date string[p87] representing the user's selection. User agents should allow the user to set the value[p626] to the empty string.

Constraint validation: While the user interface describes input that the user agent cannot convert to a valid date string[p87], the control is suffering from bad input[p652].

Note

See the introduction section[p531] for a discussion of the difference between the input format and submission format for date, time, and number form controls, and the implementation notes[p569] regarding localization of form controls.

The `value`[p543] attribute, if specified and not empty, must have a value that is a valid date string[p87].

The value sanitization algorithm[p543] is as follows: If the value[p626] of the element is not a valid date string[p87], then set it to the empty string instead.

The `min`[p574] attribute, if specified, must have a value that is a valid date string[p87]. The `max`[p574] attribute, if specified, must have a value that is a valid date string[p87].

The `step`[p575] attribute is expressed in days. The step scale factor[p575] is 86,400,000 (which converts the days to milliseconds, as used in the other algorithms). The default step[p575] is 1 day.

When the element is suffering from a step mismatch[p652], the user agent may round the element's value[p626] to the nearest date[p87] for which the element would not suffer from a step mismatch[p652].

The algorithm to convert a string to a number[p543], given a string *input*, is as follows: If parsing a date[p87] 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[p87], ignoring leap seconds.

The algorithm to convert a number to a string[p543], given a number *input*, is as follows: Return a valid date string[p87] that represents the date[p87] 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[p543], given a string *input*, is as follows: If parsing a date[p87] from *input* results in an error, then return an error; otherwise, return a new `Date` object[p58] representing midnight UTC on the morning of the parsed date[p87].

The algorithm to convert a `Date` object to a string[p543], given a `Date` object *input*, is as follows: Return a valid date string[p87] that represents the date[p87] current at the time represented by *input* in the UTC time zone.

¶[p55] 1

Note

The Date[p550] 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 millisecond 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[p550] state (and the 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`[p590] element and `input`[p539] elements with the Number[p556] state.

Bookkeeping details

- The following common `input`[p539] element content attributes, IDL attributes, and methods apply[p542] to the element: `autocomplete`[p633], `list`[p576], `max`[p574], `min`[p574], `readonly`[p570], `required`[p571], and `step`[p575] content attributes; `list`[p582], `value`[p580], `valueAsDate`[p581], and `valueAsNumber`[p581] IDL attributes; `select()`[p648], `stepDown()`[p581], and `stepUp()`[p581] methods.
- The `value`[p580] IDL attribute is in mode value[p580].
- The `input` and `change`[p1575] events apply[p542].
- The following content attributes must not be specified and do not apply[p542] to the element: `accept`[p563], `alpha`[p559], `alt`[p567], `checked`[p543], `colorspace`[p560],

dirname, formaction, formenctype, formmethod, formnovalidate, formtarget, height, maxlength, minlength, multiple, pattern, placeholder, popovertarget, popovertargetaction, size, src, and width.

- The following IDL attributes and methods do not apply to the element: checked, selectionStart, selectionEnd, and selectionDirection IDL attributes; setRangeText(), and setSelectionRange() methods.

4.10.5.1.8 Month state (type=month)

When an input element's type attribute is in the Month state, the rules in this section apply.

The input element represents a control for setting the element's value to a string representing a specific month.

If the element is mutable, the user agent should allow the user to change the month represented by its value, as obtained by parsing a month from it. User agents must not allow the user to set the value to a non-empty string that is not a valid month string. If the user agent provides a user interface for selecting a month, then the value must be set to a valid month string representing the user's selection. User agents should allow the user to set the value to the empty string.

Constraint validation: While the user interface describes input that the user agent cannot convert to a valid month string, the control is suffering from bad input.

Note

See 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 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, alpha, alt, checked, colorspace, dirname, formaction, formenctype, formmethod, formnovalidate, formtarget, height, maxlength, minlength, multiple, pattern, placeholder, popovertarget, popovertargetaction, 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) §[p553]

When an input[p539] element's type[p541] attribute is in the Week[p553] state, the rules in this section apply.

The input[p539] element represents[p148] a control for setting the element's value[p626] to a string representing a specific week[p93].

If the element is mutable[p626], the user agent should allow the user to change the week[p93] represented by its value[p626], as obtained by parsing a week[p93] from it. User agents must not allow the user to set the value[p626] to a non-empty string that is not a valid week string[p93]. If the user agent provides a user interface for selecting a week[p93], then the value[p626] must be set to a valid week string[p93] representing the user's selection. User agents should allow the user to set the value[p626] to the empty string.

Constraint validation: While the user interface describes input that the user agent cannot convert to a valid week string[p93], the control is suffering from bad input[p652].

Note

See the introduction section[p531] for a discussion of the difference between the input format and submission format for date, time, and number form controls, and the implementation notes[p569] regarding localization of form controls.

The value[p543] attribute, if specified and not empty, must have a value that is a valid week string[p93].

The value sanitization algorithm[p543] is as follows: If the value[p626] of the element is not a valid week string[p93], then set it to the empty string instead.

The min[p574] attribute, if specified, must have a value that is a valid week string[p93]. The max[p574] attribute, if specified, must have a value that is a valid week string[p93].

The step[p575] attribute is expressed in weeks. The step scale factor[p575] is 604,800,000 (which converts the weeks to milliseconds, as used in the other algorithms). The default step[p575] is 1 week. The default step base[p575] is −259,200,000 (the start of week 1970-W01).

When the element is suffering from a step mismatch[p652], the user agent may round the element's value[p626] to the nearest week[p93] for which the element would not suffer from a step mismatch[p652].

The algorithm to convert a string to a number[p543], given a string *input*, is as follows: If parsing a week string[p93] 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[p93], ignoring leap seconds.

The algorithm to convert a number to a string[p543], given a number *input*, is as follows: Return a valid week string[p93] that represents the week[p93] 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[p543], given a string *input*, is as follows: If parsing a week[p93] from *input* results in an error, then return an error; otherwise, return a new `Date` object[p58] representing midnight UTC on the morning of the Monday of the parsed week[p93].

The algorithm to convert a `Date` object to a string[p543], given a `Date` object *input*, is as follows: Return a valid week string[p93] that represents the week[p93] current at the time represented by *input* in the UTC time zone.

Bookkeeping details

- The following common input[p539] element content attributes, IDL attributes, and methods apply[p542] to the element: autocomplete[p633], list[p576], max[p574], min[p574], readonly[p570], required[p571], and step[p575] content attributes; list[p582], value[p580], valueAsDate[p581], and valueAsNumber[p581] IDL attributes; select()[p648], stepDown()[p581], and stepUp()[p581] methods.
- The value[p580] IDL attribute is in mode value[p580].
- The input and change[p1575] events apply[p542].
- The following content attributes must not be specified and do not apply[p542] to the element: accept[p563], alpha[p559], alt[p567], checked[p543], colorspace[p560], dirname[p629], formaction[p631], formenctype[p632], formmethod[p631], formnovalidate[p632], formtarget[p632], height[p495], maxlength[p569], minlength[p569], multiple[p572], pattern[p573], placeholder[p578], popovertarget[p935], popovertargetaction[p935], size[p570], src[p566], and width[p495].
- The following IDL attributes and methods do not apply[p542] to the element: checked[p581], files[p581], selectionStart[p648], selectionEnd[p649], and selectionDirection[p649] IDL attributes; setRangeText()[p650], and setSelectionRange()[p649] methods.

4.10.5.1.10 Time state (type=time) §[p55]

When an input[p539] element's type[p541] attribute is in the Time[p554] state, the rules in this section apply.

The input[p539] element represents[p148] a control for setting the element's value[p626] to a string representing a specific time[p88].

If the element is mutable[p626], the user agent should allow the user to change the time[p88] represented by its value[p626], as obtained by parsing a time[p89] from it. User agents must not allow the user to set the value[p626] to a non-empty string that is not a valid time string[p88]. If the user agent provides a user interface for selecting a time[p88], then the value[p626] must be set to a valid time string[p88] representing the user's selection. User agents should allow the user to set the value[p626] to the empty string.

Constraint validation: While the user interface describes input that the user agent cannot convert to a valid time string[p88], the control is suffering from bad input[p652].

Note

See the introduction section[p531] for a discussion of the difference between the input format and submission format for date, time, and number form controls, and the implementation notes[p569] regarding localization of form controls.

The value[p543] attribute, if specified and not empty, must have a value that is a valid time string[p88].

The value sanitization algorithm[p543] is as follows: If the value[p626] of the element is not a valid time string[p88], then set it to the empty string instead.

The form control has a periodic domain[p574].

The min[p574] attribute, if specified, must have a value that is a valid time string[p88]. The max[p574] attribute, if specified, must have a value that is a valid time string[p88].

The step[p575] attribute is expressed in seconds. The step scale factor[p575] is 1000 (which converts the seconds to milliseconds, as used in the other algorithms). The default step[p575] is 60 seconds.

When the element is suffering from a step mismatch[p652], the user agent may round the element's value[p626] to the nearest time[p88] for which the element would not suffer from a step mismatch[p652].

The algorithm to convert a string to a number[p543], given a string *input*, is as follows: If parsing a time[p89] from *input* results in an error, then return an error; otherwise, return the number of milliseconds elapsed from midnight to the parsed time[p88] on a day with no time changes.

The algorithm to convert a number to a string[p543], given a number *input*, is as follows: Return a valid time string[p88] that represents the time[p88] that is *input* milliseconds after midnight on a day with no time changes.

The algorithm to convert a string to a Date object[p543], given a string *input*, is as follows: If parsing a time[p89] from *input* results in an error, then return an error; otherwise, return a new Date object[p58] representing the parsed time[p88] in UTC on 1970-01-01.

The algorithm to convert a Date object to a string[p543], given a Date object *input*, is as follows: Return a valid time string[p88] that represents the UTC time[p88] component that is represented by *input*.

Bookkeeping details

- The following common input[p539] element content attributes, IDL attributes, and methods apply[p542] to the element: autocomplete[p633], list[p576], max[p574], min[p574], readonly[p570], required[p571], and step[p575] content attributes; list[p582], value[p580], valueAsDate[p581], and valueAsNumber[p581] IDL attributes; select()[p648], stepDown()[p581], and stepUp()[p581] methods.
- The value[p580] IDL attribute is in mode value[p580].
- The input and change[p1575] events apply[p542].
- The following content attributes must not be specified and do not apply[p542] to the element: accept[p563], alpha[p559], alt[p567], checked[p543], colorspace[p560], dirname[p629], formaction[p631], formenctype[p632], formmethod[p631], formnovalidate[p632], formtarget[p632], height[p495], maxlength[p569], minlength[p569], multiple[p572], pattern[p573], placeholder[p578], popovertarget[p935], popovertargetaction[p935], size[p570], src[p566], and width[p495].
- The following IDL attributes and methods do not apply[p542] to the element: checked[p581], files[p581], selectionStart[p648], selectionEnd[p649], and selectionDirection[p649] IDL attributes; setRangeText()[p650], and setSelectionRange()[p649] methods.

4.10.5.1.11 Local Date and Time state (type=datetime-local) §[p55]

When an input[p539] element's type[p541] attribute is in the Local Date and Time[p554] state, the rules in this section apply.

The input element represents a control for setting the element's value to a string representing a local date and time, with no time-zone offset information.

If the element is mutable, the user agent should allow the user to change the date and time represented by its value, as obtained by parsing a date and time from it. User agents must not allow the user to set the value to a non-empty string that is not a valid normalized local date and time string. If the user agent provides a user interface for selecting a local date and time, then the value must be set to a valid normalized local date and time string representing the user's selection. User agents should allow the user to set the value to the empty string.

Constraint validation: While the user interface describes input that the user agent cannot convert to a valid normalized local date and 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 local date and time string.

The value sanitization algorithm is as follows: If the value of the element is a valid local date and time string, then set it to a valid normalized local date and time string representing the same date and time; otherwise, set it to the empty string instead.

The min attribute, if specified, must have a value that is a valid local date and time string. The max attribute, if specified, must have a value that is a valid local date and 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 local date and time for which the element would not suffer from a step mismatch.

The algorithm to convert a string to a number, given a string *input*, is as follows: If parsing a date and time from *input* results in an error, then return an error; otherwise, return the number of milliseconds elapsed from midnight on the morning of 1970-01-01 (the time represented by the value "`1970-01-01T00:00:00.0`") to the parsed local date and time, ignoring leap seconds.

The algorithm to convert a number to a string, given a number *input*, is as follows: Return a valid normalized local date and time string that represents the date and time that is *input* milliseconds after midnight on the morning of 1970-01-01 (the time represented by the value "`1970-01-01T00:00:00.0`").

Note

See the note on historical dates in the Date state section.

Bookkeeping details

- The following common input element content attributes, IDL attributes, and methods apply to the element: autocomplete, list, max, min, readonly, required, and step content attributes; list, value, and valueAsNumber IDL attributes; select(), stepDown(), and stepUp() methods.
- The value IDL attribute is in mode value.
- The input and change events apply.
- The following content attributes must not be specified and do not apply to the element: accept, alpha, alt, checked, colorspace, dirname, formaction, formenctype, formmethod, formnovalidate, formtarget, height, maxlength, minlength, multiple, pattern, placeholder, popovertarget, popovertargetaction, size, src, and width.
- The following IDL attributes and methods do not apply to the element: checked, files, selectionStart, selectionEnd, selectionDirection, and valueAsDate IDL attributes; setRangeText(), and setSelectionRange() methods.

Example

The following example shows part of a flight booking application. The application uses an input element with its type attribute set to datetime-local, and it then interprets the given date and time in the time zone of the selected airport.

```
<fieldset>
 <legend>Destination</legend>
 <p><label>Airport: <input type=text name=to list=airports></label></p>
```

```
 <p><label>Departure time: <input type=datetime-local name=totime step=3600></label></p>
</fieldset>
<datalist id=airports>
 <option value=ATL label="Atlanta">
 <option value=MEM label="Memphis">
 <option value=LHR label="London Heathrow">
 <option value=LAX label="Los Angeles">
 <option value=FRA label="Frankfurt">
</datalist>
```

4.10.5.1.12 Number state (`type=number`) §[p556]

When an `input`[p539] element's `type`[p541] attribute is in the Number[p556] state, the rules in this section apply.

The `input`[p539] element represents[p148] a control for setting the element's value[p626] to a string representing a number.

If the element is mutable[p626], the user agent should allow the user to change the number represented by its value[p626], as obtained from applying the rules for parsing floating-point number values[p82] to it. User agents must not allow the user to set the value[p626] to a non-empty string that is not a valid floating-point number[p81]. If the user agent provides a user interface for selecting a number, then the value[p626] must be set to the best representation of the number representing the user's selection as a floating-point number[p82]. User agents should allow the user to set the value[p626] to the empty string.

Constraint validation: While the user interface describes input that the user agent cannot convert to a valid floating-point number[p81], the control is suffering from bad input[p652].

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`[p543] attribute, if specified and not empty, must have a value that is a valid floating-point number[p81].

The value sanitization algorithm[p543] is as follows: If the value[p626] of the element is not a valid floating-point number[p81], then set it to the empty string instead.

The `min`[p574] attribute, if specified, must have a value that is a valid floating-point number[p81]. The `max`[p574] attribute, if specified, must have a value that is a valid floating-point number[p81].

The step scale factor[p575] is 1. The default step[p575] is 1 (allowing only integers to be selected by the user, unless the step base[p575] has a non-integer value).

When the element is suffering from a step mismatch[p652], the user agent may round the element's value[p626] to the nearest number for which the element would not suffer from a step mismatch[p652]. 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[p543], given a string *input*, is as follows: If applying the rules for parsing floating-point number values[p82] to *input* results in an error, then return an error; otherwise, return the resulting number.

The algorithm to convert a number to a string[p543], given a number *input*, is as follows: Return a valid floating-point number[p81] that represents *input*.

Bookkeeping details

- The following common `input`[p539] element content attributes, IDL attributes, and methods apply[p542] to the element: `autocomplete`[p633], `list`[p576], `max`[p574], `min`[p574], `placeholder`[p578], `readonly`[p570], `required`[p571], and `step`[p575] content attributes; `list`[p582], `value`[p580], and `valueAsNumber`[p581] IDL attributes; `select()`[p648], `stepDown()`[p581], and `stepUp()`[p581] methods.
- The `value`[p580] IDL attribute is in mode value[p580].

- The input and change events apply.
- The following content attributes must not be specified and do not apply to the element: accept, alpha, alt, checked, colorspace, dirname, formaction, formenctype, formmethod, formnovalidate, formtarget, height, maxlength, minlength, multiple, pattern, popovertarget, popovertargetaction, size, src, and width.
- The following IDL attributes and methods do not apply to the element: checked, files, selectionStart, selectionEnd, selectionDirection, and valueAsDate IDL attributes; setRangeText() and setSelectionRange() methods.

Example

Here is an example of using a numeric input control:

```
<label>How much do you want to charge? $<input type=number min=0 step=0.01 name=price></label>
```

As described above, a user agent might support numeric input in the user's local format, converting it to the format required for submission as described above. This might include handling grouping separators (as in "872,000,000,000") and various decimal separators (such as "3,99" vs "3.99") or using local digits (such as those in Arabic, Devanagari, Persian, and Thai).

Note

The `type=number` state is not appropriate for input that happens to only consist of numbers but isn't strictly speaking a number. For example, it would be inappropriate for credit card numbers or US postal codes. A simple way of determining whether to use `type=number` is to consider whether it would make sense for the input control to have a spinbox interface (e.g. with "up" and "down" arrows). Getting a credit card number wrong by 1 in the last digit isn't a minor mistake, it's as wrong as getting every digit incorrect. So it would not make sense for the user to select a credit card number using "up" and "down" buttons. When a spinbox interface is not appropriate, `type=text` is probably the right choice (possibly with an inputmode or pattern attribute).

4.10.5.1.13 Range state (`type=range`)

When an input element's type attribute is in the Range state, the rules in this section apply.

The input element represents a control for setting the element's value to a string representing a number, but with the caveat that the exact value is not important, letting UAs provide a simpler interface than they do for the Number state.

If the element is mutable, the user agent should allow the user to change the number represented by its value, as obtained from applying the rules for parsing floating-point number values to it. User agents must not allow the user to set the value to a string that is not a valid floating-point number. If the user agent provides a user interface for selecting a number, then the value must be set to a best representation of the number representing the user's selection as a floating-point number. User agents must not allow the user to set the value to the empty string.

Constraint validation: While the user interface describes input that the user agent cannot convert to a valid floating-point number, the control is suffering from bad input.

The value attribute, if specified, must have a value that is a valid floating-point number.

The value sanitization algorithm is as follows: If the value of the element is not a valid floating-point number, then set it to the best representation, as a floating-point number, of the default value.

The **default value** is the minimum plus half the difference between the minimum and the maximum, unless the maximum is less than the minimum, in which case the default value is the minimum.

When the element is suffering from an underflow, the user agent must set the element's value to the best representation, as a floating-point number, of the minimum.

When the element is suffering from an overflow, if the maximum is not less than the minimum, the user agent must set the element's value to a valid floating-point number that represents the maximum.

When the element is suffering from a step mismatch, the user agent must round the element's value to the nearest number for which the element would not suffer from a step mismatch, and which is greater than or equal to the minimum, and, if the maximum is not less than the minimum, which is less than or equal to the maximum, if there is a number that matches these constraints. If two numbers match these constraints, then user agents must use the one nearest to positive infinity.

Example

For example, the markup `<input type="range" min=0 max=100 step=20 value=50>` results in a range control whose initial value is 60.

Example

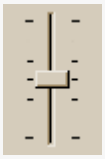
Here is an example of a range control using an autocomplete list with the `list`[p576] attribute. This could be useful if there are values along the full range of the control that are especially important, such as preconfigured light levels or typical speed limits in a range control used as a speed control. The following markup fragment:

```
<input type="range" min="-100" max="100" value="0" step="10" name="power" list="powers">
<datalist id="powers">
<option value="0">
<option value="-30">
<option value="30">
 <option value="++50">
</datalist>
```

...with the following style sheet applied:

```
input { writing-mode: vertical-lr; 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`[p575] 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.

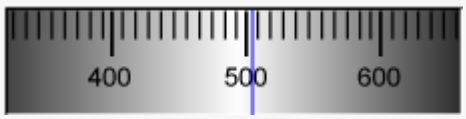
Note also how the invalid value `++50` was ignored.

Example

For another example, consider the following markup fragment:

```
<input name=x type=range min=100 max=700 step=9.09090909 value=509.090909>
```

A user agent could display in a variety of ways, for instance:



Or, alternatively, for instance:

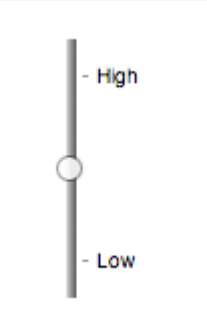
The user agent could pick which one to display based on the dimensions given in the style sheet. This would allow it to maintain the same resolution for the tick marks, despite the differences in width.

Example

Finally, here is an example of a range control with two labeled values:

```
<input type="range" name="a" list="a-values">
<datalist id="a-values">
<option value="10" label="Low">
<option value="90" label="High">
</datalist>
```

With styles that make the control draw vertically, it might look as follows:



Note

In this state, the range and step constraints are enforced even during user input, and there is no way to set the value to the empty string.

The min[p574] attribute, if specified, must have a value that is a valid floating-point number[p81]. The default minimum[p574] is 0. The max[p574] attribute, if specified, must have a value that is a valid floating-point number[p81]. The default maximum[p574] is 100.

The step scale factor[p575] is 1. The default step[p575] is 1 (allowing only integers, unless the min[p574] attribute has a non-integer value).

The algorithm to convert a string to a number[p543], given a string *input*, is as follows: If applying the rules for parsing floating-point number values[p82] to *input* results in an error, then return an error; otherwise, return the resulting number.

The algorithm to convert a number to a string[p543], given a number *input*, is as follows: Return the best representation, as a floating-point number[p82], of *input*.

Bookkeeping details

- The following common input[p539] element content attributes, IDL attributes, and methods apply[p542] to the element: autocomplete[p633], list[p576], max[p574], min[p574], and step[p575] content attributes; list[p582], value[p580], and valueAsNumber[p581] IDL attributes; stepDown()[p581] and stepUp()[p581] methods.
- The value[p580] IDL attribute is in mode value[p580].
- The input and change[p1575] events apply[p542].
- The following content attributes must not be specified and do not apply[p542] to the element: accept[p563], alpha[p559], alt[p567], checked[p543], colorspace[p560], dirname[p629], formaction[p631], formenctype[p632], formmethod[p631], formnovalidate[p632], formtarget[p632], height[p495], maxlength[p569], minlength[p569], multiple[p572], pattern[p573], placeholder[p578], popovertarget[p935], popovertargetaction[p935], readonly[p570], required[p571], size[p570], src[p566], and width[p495].
- The following IDL attributes and methods do not apply[p542] to the element: checked[p581], files[p581], selectionStart[p648], selectionEnd[p649], selectionDirection[p649], and valueAsDate[p581] IDL attributes; select()[p648], setRangeText()[p650], and setSelectionRange()[p649] methods.

✓ MDN

4.10.5.1.14 Color state (`type=color`) §[p55 9]

When an input[p539] element's type[p541] attribute is in the Color[p559] state, the rules in this section apply.

The input[p539] element represents[p148] a color well control, for setting the element's value[p626] to a string representing the serialization of a CSS color.

Note

In this state, there is always a CSS color picked, and there is no way for the end user to set the value to the empty string.

The **alpha** attribute is a boolean attribute[p79]. If present, it indicates the CSS color's alpha component can be manipulated by the end

user and does not have to be fully opaque.

The **colorspace** attribute indicates the color space of the serialized CSS color. It also hints at the desired user interface for selecting a CSS color. It is an enumerated attribute[p80] with the following keywords and states:

Keyword	State	Brief description
limited-srgb	Limited sRGB	The CSS color is converted to the 'srgb' color space and limited to 8-bits per component, e.g., "#123456" or "color(srgb 0 1 0 / 0.5)".
display-p3	Display P3	The CSS color is converted to the 'display-p3' color space, e.g., "color(display-p3 1.84 -0.19 0.72 / 0.6)".

The attribute's *missing value default*[p80] and *invalid value default*[p80] are both the Limited sRGB[p560] state.

Whenever the element's alpha[p559] or colorspace[p560] attributes are changed, the user agent must run update a color well control color[p560] given the element.

If the element is mutable[p626], the user agent should allow the user to change the color represented by its value[p626], as obtained from parsing it. User agents must not allow the user to set the value[p626] to a string that running update a color well control color[p560] for the element would not set it to. If the user agent provides a user interface for selecting a CSS color, then the value[p626] must be set to the result of serializing a color well control color[p560] given the element and the end user's selection.

The input activation behavior[p544] for such an element *element* is to show the picker, if applicable[p582], for *element*.

Constraint validation: While the element's value[p626] is not the empty string and parsing it returns failure, the control is suffering from bad input[p652].

The value[p543] attribute, if specified and not the empty string, must have a value that is a CSS color.

The value sanitization algorithm[p543] is as follows: Run update a color well control color[p560] for the element.

To **update a color well control color**, given an element *element*:

1. Assert: *element* is an input[p539] element whose type[p541] attribute is in the Color[p559] state.
2. Let *value* be the result of running these steps:
    1. If *element*'s dirty value flag[p626] is false, then return the result of getting an attribute by namespace and local name given null, "value", and *element*.
    2. Return *element*'s value[p626].
3. Let *color* be the result of parsing *value*.
4. If *color* is failure, then set *color* to opaque black.
5. Set *element*'s value[p626] to the result of serializing a color well control color[p560] given *element* and *color*.

To **serialize a color well control color**, given an element *element* and a CSS color *color*:

1. Assert: *element* is an input[p539] element whose type[p541] attribute is in the Color[p559] state.
2. Let *htmlCompatible* be false.
3. If *element*'s alpha[p559] attribute is not specified, then set *color*'s alpha component to be fully opaque.
4. If *element*'s colorspace[p560] attribute is in the Limited sRGB[p560] state:
    1. Set *color* to *color* converted to the 'srgb' color space.
    2. Round each of *color*'s components so they are in the range 0 to 255, inclusive. Components are to be rounded towards +∞.
    3. If *element*'s alpha[p559] attribute is not specified, then set *htmlCompatible* to true.

Note

This intentionally uses a different serialization path for compatibility with an earlier version of the color well control.

4. Otherwise, set *color* to *color* converted using the 'color()' function.

5. Otherwise:

    1. Assert: *element*'s colorspace[p560] attribute is in the Display P3[p560] state.

    2. Set *color* to *color* converted to the 'display-p3' color space.

6. Return the result of serializing *color*. If *htmlCompatible* is true, then do so with HTML-compatible serialization requested.

Bookkeeping details

- The following common input[p539] element content attributes and IDL attributes apply[p542] to the element: alpha[p559], autocomplete[p633], colorspace[p560], and list[p576] content attributes; list[p582] and value[p580] IDL attributes; select()[p648] method.
- The value[p580] IDL attribute is in mode value[p580].
- The input and change[p1575] events apply[p542].
- The following content attributes must not be specified and do not apply[p542] to the element: accept[p563], alt[p567], checked[p543], dirname[p629], formaction[p631], formenctype[p632], formmethod[p631], formnovalidate[p632], formtarget[p632], height[p495], max[p574], maxlength[p569], min[p574], minlength[p569], multiple[p572], pattern[p573], placeholder[p578], popovertarget[p935], popovertargetaction[p935], readonly[p570], required[p571], size[p570], src[p566], step[p575], and width[p495].
- The following IDL attributes and methods do not apply[p542] to the element: checked[p581], files[p581], selectionStart[p648], selectionEnd[p649], selectionDirection[p649], valueAsDate[p581] and, valueAsNumber[p581] IDL attributes; setRangeText()[p650], setSelectionRange()[p649], stepDown()[p581], and stepUp()[p581] methods.

✔ MDN

4.10.5.1.15 Checkbox state (type=checkbox) §[p56 1]

When an input[p539] element's type[p541] attribute is in the Checkbox[p561] state, the rules in this section apply.

The input[p539] element represents[p148] a two-state control that represents the element's checkedness[p626] state. If the element's checkedness[p626] state is true, the control represents a positive selection, and if it is false, a negative selection. If the element's indeterminateness[p545] is true, then the control's selection should be obscured.

The input activation behavior[p544] 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[p1575] at the element with the bubbles attribute initialized to true.

Constraint validation: If the element is *required*[p571] and its checkedness[p626] is false, then the element is suffering from being missing[p651].

For web developers (non-normative)

***input*.indeterminate[p545] [= *value*]**

When set, overrides the rendering of checkbox[p561] controls so that the current value is not visible.

Bookkeeping details

- The following common input[p539] element content attributes and IDL attributes apply[p542] to the element: checked[p543], and required[p571] content attributes; checked[p581] and value[p580] IDL attributes.
- The value[p580] IDL attribute is in mode default/on[p580].
- The input and change[p1575] events apply[p542].
- The following content attributes must not be specified and do not apply[p542] to the element: accept[p563], alpha[p559], alt[p567], autocomplete[p633], colorspace[p560], dirname[p629], formaction[p631], formenctype[p632], formmethod[p631], formnovalidate[p632], formtarget[p632], height[p495], list[p576], max[p574], maxlength[p569], min[p574], minlength[p569], multiple[p572], pattern[p573], placeholder[p578], popovertarget[p935], popovertargetaction[p935], readonly[p570], size[p570], src[p566], step[p575], and width[p495].
- The following IDL attributes and methods do not apply[p542] to the element: files[p581], list[p582], selectionStart[p648], selectionEnd[p649], selectionDirection[p649], valueAsDate[p581], and valueAsNumber[p581] IDL attributes; select()[p648], setRangeText()[p650], setSelectionRange()[p649], stepDown()[p581], and stepUp()[p581] methods.

✔ MDN

4.10.5.1.16 Radio Button state (type=radio) §[p56 1]

When an input[p539] element's type[p541] attribute is in the Radio Button[p561] state, the rules in this section apply.

The input element represents a control that, when used in conjunction with other input elements, forms a *radio button group* in which only one control can have its checkedness state set to true. If the element's checkedness state is true, the control represents the selected control in the group, and if it is false, it indicates a control in the group that is not selected.

The ***radio button group*** that contains an input element *a* also contains all the other input elements *b* that fulfill all of the following conditions:

- The input element *b*'s type attribute is in the Radio Button state.
- Either *a* and *b* have the same form owner, or they both have no form owner.
- Both *a* and *b* are in the same 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 checkedness state is true after the occurrence, the checkedness state of all the other elements in the same *radio button group* must be set to false:

- The element's checkedness state is set to true (for whatever reason).
- The element's name attribute is set, changed, or removed.
- The element's form owner changes.
- A type change is signalled for the element.
- The element becomes connected.

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 checkedness that is false, then the element is suffering from being missing.

Example

The following example, for some reason, has specified that puppers are both required and disabled:

```
<form>
 <p><label><input type="radio" name="dog-type" value="pupper" required disabled> Pupper</label>
 <p><label><input type="radio" name="dog-type" value="doggo"> Doggo</label>
 <p><button>Make your choice</button>
</form>
```

If the user tries to submit this form without first selecting "Doggo", then *both* input elements will be suffering from being missing, since an element in the radio button group is required (viz. the first element), and both of the elements in the radio button group have a false checkedness.

On the other hand, if the user selects "Doggo" and then submits the form, then neither input element will be suffering from being missing, since while one of them is required, not all of them have a false checkedness.

Note

If none of the radio buttons in a radio button group 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 element content attributes and IDL attributes apply to the element: checked and required content attributes; checked and value IDL attributes.

- The value[p580] IDL attribute is in mode default/on[p580].
- The input and change[p1575] events apply[p542].
- The following content attributes must not be specified and do not apply[p542] to the element: accept[p563], alpha[p559], alt[p567], autocomplete[p633], colorspace[p560], dirname[p629], formaction[p631], formenctype[p632], formmethod[p631], formnovalidate[p632], formtarget[p632], height[p495], list[p576], max[p574], maxlength[p569], min[p574], minlength[p569], multiple[p572], pattern[p573], placeholder[p578], popovertarget[p935], popovertargetaction[p935], readonly[p570], size[p570], src[p566], step[p575], and width[p495].
- The following IDL attributes and methods do not apply[p542] to the element: files[p581], list[p582], selectionStart[p648], selectionEnd[p649], selectionDirection[p649], valueAsDate[p581], and valueAsNumber[p581] IDL attributes; select()[p648], setRangeText()[p650], setSelectionRange()[p649], stepDown()[p581], and stepUp()[p581] methods.

4.10.5.1.17 File Upload state (`type=file`) §[p563]

When an input[p539] element's type[p541] attribute is in the File Upload[p563] state, the rules in this section apply.

The input[p539] element represents[p148] a list of **selected files**, each file consisting of a filename, a file type, and a file body (the contents of the file).

Filenames must not contain path components[p563], 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[p563] state, are those parts of filenames that are separated by U+005C REVERSE SOLIDUS character (\) characters.

Unless the multiple[p572] attribute is set, there must be no more than one file in the list of selected files[p563].

The input activation behavior[p544] for such an element *element* is to show the picker, if applicable[p582], for *element*.

If the element is mutable[p626], 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[p563] for the element.

If the element is not mutable[p626], the user agent must not allow the user to change the element's selection.

To **update the file selection** for an element *element*:

1. Queue an element task[p1195] on the user interaction task source[p1204] given *element* and the following steps:
    1. Update *element*'s selected files[p563] so that it represents the user's selection.
    2. Fire an event named input at the input[p539] element, with the bubbles and composed attributes initialized to true.
    3. Fire an event named change[p1575] at the input[p539] element, with the bubbles attribute initialized to true.

Constraint validation: If the element is *required*[p571] and the list of selected files[p563] is empty, then the element is suffering from being missing[p651].

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[p99], 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 filenames 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[p563] attribute's requirements.

¶[p564] **Example**

For historical reasons, the value[p580] IDL attribute prefixes the filename 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 filename from the value[p580] IDL attribute in a backwards-compatible way is non-trivial. The following function extracts the filename in a suitably compatible manner:

```
function extractFilename(path) {
  if (path.substr(0, 12) == "C:\\fakepath\\")
    return path.substr(12); // modern browser
  var x;
  x = path.lastIndexOf('/');
  if (x >= 0) // Unix-based path
    return path.substr(x+1);
  x = path.lastIndexOf('\\');
  if (x >= 0) // Windows-based path
    return path.substr(x+1);
  return path; // just the filename
}
```

This can be used as follows:

```
<p><input type=file name=image onchange="updateFilename(this.value)"></p>
<p>The name of the file you picked is: <span id="filename">(none)</span></p>
<script>
 function updateFilename(path) {
```

```
    var name = extractFilename(path);
    document.getElementById('filename').textContent = name;
  }
</script>
```

Bookkeeping details

- The following common `input` element content attributes and IDL attributes apply to the element: `accept`, `multiple`, and `required` content attributes; `files` and `value` IDL attributes; `select()` method.
- The `value` IDL attribute is in mode filename.
- The `input` and `change` events apply.
- The following content attributes must not be specified and do not apply to the element: `alpha`, `alt`, `autocomplete`, `checked`, `colorspace`, `dirname`, `formaction`, `formenctype`, `formmethod`, `formnovalidate`, `formtarget`, `height`, `list`, `max`, `maxlength`, `min`, `minlength`, `pattern`, `popovertarget`, `popovertargetaction`, `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`) §

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 optional value is the value of the element's `value` attribute, if there is one; otherwise null.

The element's input activation behavior given *event* is as follows:

1. If the element does not have a form owner, then return.
2. If the element's node document is not fully active, then return.
3. Submit the element's form owner from the element with *userInvolvement* set to *event*'s user navigation involvement.

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: `dirname`, `formaction`, `formenctype`, `formmethod`, `formnovalidate`, `formtarget`, `popovertarget`, and `popovertargetaction` 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`, `alpha`, `alt`, `autocomplete`, `checked`, `colorspace`, `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.

4.10.5.1.19 Image Button state (type=image) §

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 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, run these steps:

1. Let *url* be the result of encoding-parsing a URL given the src attribute's value, relative to the element's node document.
2. If *url* is failure, then return.
3. Let *request* be a new request whose URL is *url*, client is the element's node document's relevant settings object, destination is "image", initiator type is "input", credentials mode is "include", and whose use-URL-credentials flag is set.
4. Fetch *request*, with *processResponseEndOfBody* set to the following steps given response *response*:
    1. If the download was successful and the image is *available*, queue an element task on the user interaction task source given the input element to fire an event named load at the input element.
    2. 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, then queue an element task on the user interaction task source given the input element to fire an event named error on the input element.

Fetching the image must delay the load event of the element's node document until the task that is queued by the networking task source once the resource has been fetched (defined below) has been run.

If the image was successfully obtained, with no network errors, and the image's type is a supported image type, and the image is a valid image of that type, then the image is said to be ***available***. If this is true before the image is completely downloaded, each task that is queued by the networking task source while the image is being fetched must update the presentation of the image appropriately.

The user agent should apply the image sniffing rules to determine the type of the image, with the image's associated Content-Type headers giving the *official type*. If these rules are not applied, then the type of the image must be the type given by the image's associated Content-Type headers.

User agents must not support non-image resources with the input element. User agents must not run executable code embedded in the image resource. User agents must only display the first page of a multipage resource. User agents must not allow the resource to act in an interactive fashion, but should honor any animation in the resource.

The **alt** attribute provides the textual label for the button for users and user agents who cannot use the image. The alt[p567] 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[p539] element supports dimension attributes[p495].

If the src[p566] attribute is set, and the image is *available*[p566] and the user agent is configured to display that image, then the element represents[p148] a control for selecting a coordinate[p567] from the image specified by the src[p566] attribute. In that case, if the element is mutable[p626], the user agent should allow the user to select this coordinate[p567].

Otherwise, the element represents[p148] a submit button whose label is given by the value of the alt[p567] attribute.

The element's input activation behavior[p544] given *event* is as follows:

1. If the element does not have a form owner[p626], then return.
2. If the element's node document is not fully active[p1051], then return.
3. If the user activated the control while explicitly selecting a coordinate, then set the element's selected coordinate[p567] to that coordinate.

   Note

   This is only possible under the conditions outlined above, when the element represents[p148] a control for selecting such a coordinate. Even then, the user might activate the control without explicitly selecting a coordinate.

4. Submit[p658] the element's form owner[p626] from the element with *userInvolvement*[p658] set to *event*'s user navigation involvement[p1062].

The element's **selected coordinate** consists of an *x*-component and a *y*-component. It is initially (0, 0). The coordinates represent the position relative to the edge of the element's 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[p80] representing a number x in the range $-(border_{left}+padding_{left}) \le x \le width+border_{right}+padding_{right}$, 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[p80] representing a number y in the range $-(border_{top}+padding_{top}) \le y \le height+border_{bottom}+padding_{bottom}$, 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[p631], formenctype[p632], formmethod[p631], formnovalidate[p632], and formtarget[p632] attributes are attributes for form submission[p631].

For web developers (non-normative)

***image*.width[p545] [= *value*]**

***image*.height[p545] [= *value*]**

These attributes return the rendered dimensions of the image. When the image is not being rendered, they return the width[p495] and height[p495] content attributes' values, defaulting to the image's natural dimensions, if any; otherwise 0.

They can be set, to change the corresponding content attributes.

Bookkeeping details

- The following common input[p539] element content attributes and IDL attributes apply[p542] to the element: alt[p567], formaction[p631], formenctype[p632], formmethod[p631], formnovalidate[p632], formtarget[p632], height[p495], popovertarget[p935], popovertargetaction[p935], src[p566], and width[p495] content attributes; value[p580] IDL attribute.
- The value[p580] IDL attribute is in mode default[p580].
- The following content attributes must not be specified and do not apply[p542] to the element: accept[p563], alpha[p559], autocomplete[p633], checked[p543], colorspace[p560], dirname[p629], list[p576], max[p574], maxlength[p569], min[p574], minlength[p569], multiple[p572], pattern[p573], placeholder[p578], readonly[p570], required[p571], size[p570], and

step[p575].

- The element's value[p543] attribute must be omitted.
- The following IDL attributes and methods do not apply[p542] to the element: checked[p581], files[p581], list[p582], selectionStart[p648], selectionEnd[p649], selectionDirection[p649], valueAsDate[p581], and valueAsNumber[p581] IDL attributes; select()[p648], setRangeText()[p650], setSelectionRange()[p649], stepDown()[p581], and stepUp()[p581] methods.
- The input and change[p1575] events do not apply[p542].

Note

Many aspects of this state's behavior are similar to the behavior of the img[p359] element. Readers are encouraged to read that section, where many of the same requirements are described in more detail.

Example

Take the following form:

```
<form action="process.cgi">
 <input type=image src=map.png name=where alt="Show location list">
</form>
```

If the user clicked on the image at coordinate (127,40) then the URL used to submit the form would be "process.cgi?where.x=127&where.y=40".

(In this example, it's assumed that for users who don't see the map, and who instead just see a button labeled "Show location list", clicking the button will cause the server to show a list of locations to pick from instead of the map.)

4.10.5.1.20 Reset Button state (type=reset) §[p568]

When an input[p539] element's type[p541] attribute is in the Reset Button[p568] state, the rules in this section apply.

The input[p539] element represents[p148] a button that, when activated, resets the form. If the element has a value[p543] 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[p532].

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[p544] is as follows:

1. If the element does not have a form owner[p626], then return.
2. If the element's node document is not fully active[p1051], then return.
3. Reset[p666] the form owner[p626] from the element.

Constraint validation: The element is barred from constraint validation[p651].

Bookkeeping details

- The value[p580] IDL attribute applies[p542] to this element and is in mode default[p580].
- The following common input[p539] element content attributes apply[p542] to the element: popovertarget[p935] and popovertargetaction[p935].
- The following content attributes must not be specified and do not apply[p542] to the element: accept[p563], alpha[p559], alt[p567], autocomplete[p633], checked[p543], colorspace[p560], dirname[p629], formaction[p631], formenctype[p632], formmethod[p631], formnovalidate[p632], formtarget[p632], height[p495], list[p576], max[p574], maxlength[p569], min[p574], minlength[p569], multiple[p572], pattern[p573], placeholder[p578], readonly[p570], required[p571], size[p570], src[p566], step[p575], and width[p495].
- The following IDL attributes and methods do not apply[p542] to the element: checked[p581], files[p581], list[p582], selectionStart[p648], selectionEnd[p649], selectionDirection[p649], valueAsDate[p581], and valueAsNumber[p581] IDL attributes; select()[p648], setRangeText()[p650], setSelectionRange()[p649], stepDown()[p581], and stepUp()[p581] methods.
- The input and change[p1575] events do not apply[p542].

4.10.5.1.21 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 common input element content attributes apply to the element: popovertarget and popovertargetaction.
- The following content attributes must not be specified and do not apply to the element: accept, alpha, alt, autocomplete, checked, colorspace, 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, valueAsDate, and valueAsNumber IDL attributes; select(), setRangeText(), setSelectionRange(), stepDown(), and stepUp() methods.
- The input and change events do not apply.

4.10.5.2 Implementation notes regarding localization of form controls

This section is non-normative.

The formats shown to the user in date, time, and number controls is independent of the format used for form submission.

Browsers are encouraged to use user interfaces that present dates, times, and numbers according to the conventions of either the locale implied by the input element's language or the user's preferred locale. Using the page's locale will ensure consistency with page-provided data.

Example

For example, it would be confusing to users if an American English page claimed that a Cirque De Soleil show was going to be showing on 02/03, but their browser, configured to use the British English locale, only showed the date 03/02 in the ticket purchase date picker. Using the page's locale would at least ensure that the date was presented in the same format everywhere. (There's still a risk that the user would end up arriving a month late, of course, but there's only so much that can be done about such cultural differences...)

4.10.5.3 Common input element attributes

These attributes only apply to an input element if its type attribute is in a state whose definition declares that the attribute applies. When an attribute doesn't apply to an input element, user agents must ignore the attribute, regardless of the requirements and definitions below.

4.10.5.3.1 The maxlength and minlength attributes

The **maxlength** attribute, when it applies, is a form control maxlength attribute.

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 less than or equal to 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.

```
<label>What are you doing? <input name=status maxlength=140></label>
```

Example

Here, a password is given a minimum length:

```
<p><label>Username: <input name=u required></label>
<p><label>Password: <input name=p required minlength=12></label>
```

4.10.5.3.2 The size attribute

The **size** attribute gives the number of characters that, in a visual rendering, the user agent is to allow the user to see while editing the element's value.

The size attribute, if specified, must have a value that is a valid non-negative integer greater than zero.

If the attribute is present, then its value must be parsed using the rules for parsing non-negative integers, and if the result is a number greater than zero, then the user agent should ensure that at least that many characters are visible.

The size IDL attribute is limited to only positive numbers 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).

```
<form action="products.cgi" method="post" enctype="multipart/form-data">
 <table>
  <tr> <th> Product ID <th> Product name <th> Price <th> Action
  <tr>
   <td> <input readonly="readonly" name="1.pid" value="H412">
   <td> <input required="required" name="1.pname" value="Floor lamp Ulke">
   <td> $<input required="required" type="number" min="0" step="0.01" name="1.pprice"
value="49.99">
   <td> <button formnovalidate="formnovalidate" name="action" value="delete:1">Delete</button>
```

```
  <tr>
   <td> <input readonly="readonly" name="2.pid" value="FG28">
   <td> <input required="required" name="2.pname" value="Table lamp Ulke">
   <td> $<input required="required" type="number" min="0" step="0.01" name="2.pprice"
value="24.99">
   <td> <button formnovalidate="formnovalidate" name="action" value="delete:2">Delete</button>
  <tr>
   <td> <input required="required" name="3.pid" value="" pattern="[A-Z0-9]+">
   <td> <input required="required" name="3.pname" value="">
   <td> $<input required="required" type="number" min="0" step="0.01" name="3.pprice" value="">
   <td> <button formnovalidate="formnovalidate" name="action" value="delete:3">Delete</button>
 </table>
 <p> <button formnovalidate="formnovalidate" name="action" value="add">Add</button> </p>
 <p> <button name="action" value="update">Save</button> </p>
</form>
```

4.10.5.3.4 The required[p571] attribute §[p57]

The **required** attribute is a boolean attribute[p79]. When specified, the element is ***required***.

Constraint validation: If the element is *required*[p571], and its value[p580] IDL attribute applies[p542] and is in the mode value[p580], and the element is mutable[p626], and the element's value[p626] is the empty string, then the element is suffering from being missing[p651].

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.

```
<h1>Create new account</h1>
<form action="/newaccount" method=post
      oninput="up2.setCustomValidity(up2.value != up.value ? 'Passwords do not match.' : '')">
 <p>
  <label for="username">Email address:</label>
  <input id="username" type=email required name=un>
 <p>
  <label for="password1">Password:</label>
  <input id="password1" type=password required name=up>
 <p>
  <label for="password2">Confirm password:</label>
  <input id="password2" type=password name=up2>
 <p>
  <input type=submit value="Create account">
</form>
```

Example

For radio buttons, the required[p571] attribute is satisfied if any of the radio buttons in the group[p562] is selected. Thus, in the following example, any of the radio buttons can be checked, not just the one marked as required:

```
<fieldset>
 <legend>Did the movie pass the Bechdel test?</legend>
 <p><label><input type="radio" name="bechdel" value="no-characters"> No, there are not even two
female characters in the movie. </label>
 <p><label><input type="radio" name="bechdel" value="no-names"> No, the female characters never
talk to each other. </label>
 <p><label><input type="radio" name="bechdel" value="no-topic"> No, when female characters talk to
each other it's always about a male character. </label>
 <p><label><input type="radio" name="bechdel" value="yes" required> Yes. </label>
```

```
 <p><label><input type="radio" name="bechdel" value="unknown"> I don't know. </label>
</fieldset>
```

To avoid confusion as to whether a radio button group[p562] 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`[p572] attribute §[p57 2]

The **`multiple`** attribute is a boolean attribute[p79] 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.

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`[p598] element.

Example

The following extract shows how an email client's "Attachments" field could accept multiple files for upload.

```
<label>Attachments: <input type=file multiple name=att></label>
```

4.10.5.3.6 The pattern[p573] attribute §[p57 3]

The **pattern** attribute specifies a regular expression against which the control's value[p626], or, when the multiple[p572] attribute applies[p542] and is set, the control's values[p626], are to be checked.

If specified, the attribute's value must match the JavaScript *Pattern*`[+UnicodeSetsMode, +N]` production.

The **compiled pattern regular expression** of an input[p539] element, if it exists, is a JavaScript RegExp object. It is determined as follows:

1. If the element does not have a pattern[p573] attribute specified, then return nothing. The element has no compiled pattern regular expression[p573].
2. Let *pattern* be the value of the pattern[p573] attribute of the element.
3. Let *regexpCompletion* be RegExpCreate(*pattern*, "`v`").
4. If *regexpCompletion* is an abrupt completion, then return nothing. The element has no compiled pattern regular expression[p573].

   Note

   User agents are encouraged to log this error in a developer console, to aid debugging.
5. Let *anchoredPattern* be the string "`^(?:`", followed by *pattern*, followed by "`)$`".
6. Return ! RegExpCreate(*anchoredPattern*, "`v`").

Note

The reasoning behind these steps, instead of just using the value of the pattern[p573] attribute directly, is twofold. First, we want to ensure that when matched against a string, the regular expression's start is anchored to the start of the string and its end to the end of the string. Second, we want to ensure that the regular expression is valid in standalone form, instead of only becoming valid after being surrounded by the "`^(?:`" and "`)$`" anchors.

A RegExp object *regexp* **matches** a string *input*, if ! RegExpBuiltinExec(*regexp*, *input*) is not null.

Constraint validation: If the element's value[p626] is not the empty string, and either the element's multiple[p572] attribute is not specified or it does not apply[p542] to the input[p539] element given its type[p541] attribute's current state, and the element has a compiled pattern regular expression[p573] but that regular expression does not match[p573] the element's value[p626], then the element is suffering from a pattern mismatch[p651].

Constraint validation: If the element's value[p626] is not the empty string, and the element's multiple[p572] attribute is specified and applies[p542] to the input[p539] element, and the element has a compiled pattern regular expression[p573] but that regular expression does not match[p573] each of the element's values[p626], then the element is suffering from a pattern mismatch[p651].

When an input[p539] element has a pattern[p573] 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[p875].

Example

For example, the following snippet:

```
<label> Part number:
 <input pattern="[0-9][A-Z]{3}" name="part"
        title="A part number is a digit followed by three uppercase letters."/>
</label>
```

...could cause the 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`s as if an error has necessarily occurred.

4.10.5.3.7 The `min` and `max` attributes

Some form controls can have explicit constraints applied limiting the allowed range of values that the user can provide. Normally, such a range would be linear and continuous. A form control can **have a periodic domain**, however, in which case the form control's broadest possible range is finite, and authors can specify explicit ranges within it that span the boundaries.

Example

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 the element has a minimum and does not have a reversed range, and the result of applying the algorithm to convert a string to a number to the string given by the element's value is a number, and the number obtained from that algorithm is less than the minimum, the element is suffering from an underflow.

Constraint validation: When the element has a maximum and does not have a reversed range, and the result of applying the algorithm to convert a string to a number to the string given by the element's value is a number, and the number obtained from that algorithm is more than the maximum, the element is suffering from an overflow.

Constraint validation: When an element has a reversed range[p574], and the result of applying the algorithm to convert a string to a number[p543] to the string given by the element's value[p626] is a number, and the number obtained from that algorithm is more than the maximum[p574] *and* less than the minimum[p574], the element is simultaneously suffering from an underflow[p652] and suffering from an overflow[p652].

Example

The following date control limits input to dates that are before the 1980s:

```
<input name=bday type=date max="1979-12-31">
```

Example

The following number control limits input to whole numbers greater than zero:

```
<input name=quantity required="" type="number" min="1" value="1">
```

Example

The following time control limits input to those minutes that occur between 9pm and 6am, defaulting to midnight:

```
<input name="sleepStart" type=time min="21:00" max="06:00" step="60" value="00:00">
```

4.10.5.3.8 The `step`[p575] attribute §[p57 5]

The **`step`** attribute indicates the granularity that is expected (and required) of the value[p626] or values[p626], by limiting the allowed values. The section that defines the `type`[p541] 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`[p575] attribute, if specified, must either have a value that is a valid floating-point number[p81] that parses[p82] 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[p542], then there is no allowed value step[p575].
2. Otherwise, if the attribute is absent, then the allowed value step[p575] is the default step[p575] multiplied by the step scale factor[p575].
3. Otherwise, if the attribute's value is an ASCII case-insensitive match for the string "any", then there is no allowed value step[p575].
4. Otherwise, if the rules for parsing floating-point number values[p82], when they are applied to the attribute's value, return an error, zero, or a number less than zero, then the allowed value step[p575] is the default step[p575] multiplied by the step scale factor[p575].
5. Otherwise, the allowed value step[p575] is the number returned by the rules for parsing floating-point number values[p82] when they are applied to the attribute's value, multiplied by the step scale factor[p575].

The **step base** is the value returned by the following algorithm:

1. If the element has a `min`[p574] content attribute, and the result of applying the algorithm to convert a string to a number[p543] to the value of the `min`[p574] content attribute is not an error, then return that result.
2. If the element has a `value`[p543] content attribute, and the result of applying the algorithm to convert a string to a number[p543] to the value of the `value`[p543] content attribute is not an error, then return that result.
3. If a default step base[p575] is defined for this element given its `type`[p541] attribute's state, then return it.
4. Return zero.

Constraint validation: When the element has an allowed value step[p575], and the result of applying the algorithm to convert a string to a number[p543] to the string given by the element's value[p626] is a number, and that number subtracted from the step base[p575] is not an

integral multiple of the allowed value step[p575], the element is suffering from a step mismatch[p652].

Example

The following range control only accepts values in the range 0..1, and allows 256 steps in that range:

```
<input name=opacity type=range min=0 max=1 step=0.00392156863>
```

Example

The following control allows any time in the day to be selected, with any accuracy (e.g. thousandth-of-a-second accuracy or more):

```
<input name=favtime type=time step=any>
```

Normally, time controls are limited to an accuracy of one minute.

4.10.5.3.9 The `list`[p576] attribute §[p576]

The **`list`** attribute is used to identify an element that lists predefined options suggested to the user.

✔ MDN

If present, its value must be the ID of a `datalist`[p598] element in the same tree.

The **suggestions source element** is the first element in the tree in tree order to have an ID equal to the value of the `list`[p576] attribute, if that element is a `datalist`[p598] element. If there is no `list`[p576] attribute, or if there is no element with that ID, or if the first element with that ID is not a `datalist`[p598] element, then there is no suggestions source element[p576].

If there is a suggestions source element[p576], then, when the user agent is allowing the user to edit the `input`[p539] element's value[p626], the user agent should offer the suggestions represented by the suggestions source element[p576] to the user in a manner suitable for the type of control used. If appropriate, the user agent should use the suggestion's label[p603] and value[p603] to identify the suggestion to the user.

User agents are encouraged to filter the suggestions represented by the suggestions source element[p576] 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 search within both the label[p603] and value[p603] of the suggestions for matches. User agents should consider how input variations affect the matching process. Unicode normalization should be applied so that different underlying Unicode code point sequences, caused by different keyboard- or input-specific mechanisms, do not interfere with the matching process. Case variations should be ignored, which may require language-specific case mapping. For examples of these, see *Character Model for the World Wide Web: String Matching*. User agents may also provide other matching features: for illustration, a few examples include matching different forms of kana to each other (or to kanji), ignoring accents, or applying spelling correction. [CHARMODNORM][p1579]

Example

This text field allows you to choose a type of JavaScript function.

```
<input type="text" list="function-types">
<datalist id="function-types">
  <option value="function">function</option>
  <option value="async function">async function</option>
  <option value="function*">generator function</option>
  <option value="=>">arrow function</option>
  <option value="async =>">async arrow function</option>
  <option value="async function*">async generator function</option>
</datalist>
```

For user agents that follow the above suggestions, both the label[p603] and value[p603] would be shown:

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`[p572] attribute specified or if the `multiple`[p572] attribute does not apply[p542]**

When the user selects a suggestion, the `input`[p539] element's value[p626] must be set to the selected suggestion's value[p603], as if the user had written that value themself.

↪ **If the element's `type`[p541] attribute is in the Email[p548] state and the element has a `multiple`[p572] attribute specified**

When the user selects a suggestion, the user agent must either add a new entry to the `input`[p539] element's values[p626], whose value is the selected suggestion's value[p603], or change an existing entry in the `input`[p539] element's values[p626] to have the value given by the selected suggestion's value[p603], 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`[p576] attribute does not apply[p542], there is no suggestions source element[p576].

Example

This URL field offers some suggestions.

```
<label>Homepage: <input name=hp type=url list=hpurls></label>
<datalist id=hpurls>
 <option value="https://www.google.com/" label="Google">
 <option value="https://www.reddit.com/" label="Reddit">
</datalist>
```

Other URLs from the user's history might show also; this is up to the user agent.

Example

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`[p598] element with children `option`[p601] elements is enough. To prevent the values from being rendered in legacy user agents, they need to be placed inside the `value`[p603] attribute instead of inline.

```
<p>
 <label>
  Enter a breed:
```

```
  <input type="text" name="breed" list="breeds">
  <datalist id="breeds">
   <option value="Abyssinian">
   <option value="Alpaca">
   <!-- ... -->
  </datalist>
 </label>
</p>
```

However, if the values need to be shown in legacy UAs, then fallback content can be placed inside the datalist[p598] element, as follows:

```
<p>
 <label>
  Enter a breed:
  <input type="text" name="breed" list="breeds">
 </label>
 <datalist id="breeds">
  <label>
   or select one from the list:
   <select name="breed">
    <option value=""> (none selected)
    <option>Abyssinian
    <option>Alpaca
    <!-- ... -->
   </select>
  </label>
 </datalist>
</p>
```

The fallback content will only be shown in UAs that don't support datalist[p598]. The options, on the other hand, will be detected by all UAs, even though they are not children of the datalist[p598] element.

Note that if an option[p601] element used in a datalist[p598] is selected[p603], it will be selected by default by legacy UAs (because it affects the select[p590] element), but it will not have any effect on the input[p539] element in UAs that support datalist[p598].

4.10.5.3.10 The placeholder[p578] attribute §[p578]

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[p578] attribute should not be used as an alternative to a label[p536]. For a longer hint or other advisory text, the title[p164] attribute is more appropriate.

Note

These mechanisms are very similar but subtly different: the hint given by the control's label[p536] is shown at all times; the short hint given in the placeholder[p578] attribute is shown before the user enters a value; and the hint in the title[p164] 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[p626] is the empty string, especially if the control is not focused[p875].

If a user agent normally doesn't show this hint to the user when the control is focused[p875], then the user agent should nonetheless show the hint for the control if it was focused as a result of the autofocus[p887] 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[p578] attribute:

```
<fieldset>
 <legend>Mail Account</legend>
 <p><label>Name: <input type="text" name="fullname" placeholder="John Ratzenberger"></label></p>
 <p><label>Address: <input type="email" name="address" placeholder="john@example.net"></label></p>
 <p><label>Password: <input type="password" name="password"></label></p>
 <p><label>Description: <input type="text" name="desc" placeholder="My Email Account"></label></p>
</fieldset>
```

Example

In situations where the control's content has one directionality but the placeholder needs to have a different directionality, Unicode's bidirectional-algorithm formatting characters can be used in the attribute value:

```
<input name=t1 type=tel placeholder="&#x202B;1 تف الها رقم&#x202E;">
<input name=t2 type=tel placeholder="&#x202B;2 تف الها رقم&#x202E;">
```

For slightly more clarity, here's the same example using numeric character references instead of inline Arabic:

```
<input name=t1 type=tel
placeholder="&#x202B;&#1585;&#1602;&#1605; &#1575;&#1604;&#1607;&#1575;&#1578;&#1601; 1&#x202E;">
<input name=t2 type=tel
placeholder="&#x202B;&#1585;&#1602;&#1605; &#1575;&#1604;&#1607;&#1575;&#1578;&#1601; 2&#x202E;">
```

4.10.5.4 Common input[p539] element APIs §[p579]

For web developers (non-normative)

***input*.value[p580] [= *value*]**

Returns the current value[p626] of the form control.

Can be set, to change the value.

Throws an "InvalidStateError" DOMException if it is set to any value other than the empty string when the control is a file upload control.

***input*.checked[p581] [= *value*]**

Returns the current checkedness[p626] of the form control.

Can be set, to change the checkedness[p626].

***input*.files[p581] [= *files*]**

Returns a FileList object listing the selected files[p563] 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[p563] of the form control. For instance, as the result of a drag-and-drop operation.

***input*.valueAsDate[p581] [= *value*]**

Returns a Date object representing the form control's value[p626], if applicable; otherwise, returns null.

Can be set, to change the value.

Throws an "InvalidStateError" DOMException if the control isn't date- or time-based.

***input*.valueAsNumber[p581] [= *value*]**

Returns a number representing the form control's value[p626], if applicable; otherwise, returns NaN.

Can be set, to change the value. Setting this to NaN will set the underlying value to the empty string.

Throws an "InvalidStateError" DOMException if the control is neither date- or time-based nor numeric.

***input*.stepUp([*n*])**

***input*.stepDown([*n*])**

Changes the form control's value by the value given in the step attribute, multiplied by *n*. The default value for *n* is 1.

Throws "InvalidStateError" DOMException if the control is neither date- or time-based nor numeric, or if the step attribute's value is "any".

***input*.list**

Returns the datalist element indicated by the list attribute.

***input*.showPicker()**

Shows any applicable picker UI for *input*, so that the user can select a value.

If *input* does not support a picker, this method does nothing.

Throws an "InvalidStateError" DOMException if *input* is not mutable.

Throws a "NotAllowedError" DOMException if called without transient user activation.

Throws a "SecurityError" DOMException if *input* is inside a cross-origin iframe, unless *input* is in the File Upload or Color states.

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[p563] are not permitted in filenames in the list of selected files[p563], the "\fakepath\" cannot be mistaken for a path component.

The **checked** IDL attribute allows scripts to manipulate the checkedness[p626] of an input[p539] element. On getting, it must return the current checkedness[p626] of the element; and on setting, it must set the element's checkedness[p626] to the new value and set the element's dirty checkedness flag[p543] to true.

The **files** IDL attribute allows scripts to access the element's selected files[p563].

On getting, if the IDL attribute applies[p542], it must return a FileList object that represents the current selected files[p563]. The same object must be returned until the list of selected files[p563] changes. If the IDL attribute does not apply[p542], then it must instead return null. [FILEAPI][p1582]

On setting, it must run these steps:

1. If the IDL attribute does not apply[p542] or the given value is null, then return.
2. Replace the element's selected files[p563] with the given value.

The **valueAsDate** IDL attribute represents the value[p626] of the element, interpreted as a date.

On getting, if the valueAsDate[p581] attribute does not apply[p542], as defined for the input[p539] element's type[p541] attribute's current state, then return null. Otherwise, run the algorithm to convert a string to a Date object[p543] defined for that state to the element's value[p626]; if the algorithm returned a Date object, then return it, otherwise, return null.

On setting, if the valueAsDate[p581] attribute does not apply[p542], as defined for the input[p539] element's type[p541] 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[p626] of the element to the empty string; otherwise, run the algorithm to convert a Date object to a string[p543], as defined for that state, on the new value, and set the value[p626] of the element to the resulting string.

The **valueAsNumber** IDL attribute represents the value[p626] of the element, interpreted as a number.

On getting, if the valueAsNumber[p581] attribute does not apply[p542], as defined for the input[p539] element's type[p541] attribute's current state, then return a Not-a-Number (NaN) value. Otherwise, run the algorithm to convert a string to a number[p543] defined for that state to the element's value[p626]; 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[p581] attribute does not apply[p542], as defined for the input[p539] element's type[p541] 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[p626] of the element to the empty string. Otherwise, run the algorithm to convert a number to a string[p543], as defined for that state, on the new value, and set the value[p626] 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()[p581] and stepUp()[p581] methods do not apply[p542], as defined for the input[p539] element's type[p541] attribute's current state, then throw an "InvalidStateError" DOMException.
2. If the element has no allowed value step[p575], then throw an "InvalidStateError" DOMException.
3. If the element has a minimum[p574] and a maximum[p574] and the minimum[p574] is greater than the maximum[p574], then return.
4. If the element has a minimum[p574] and a maximum[p574] and there is no value greater than or equal to the element's minimum[p574] and less than or equal to the element's maximum[p574] that, when subtracted from the step base[p575], is an integral multiple of the allowed value step[p575], then return.
5. If applying the algorithm to convert a string to a number[p543] to the string given by the element's value[p626] 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[p575] is not an integral multiple of the allowed value step[p575], then set *value* to the nearest value that, when subtracted from the step base[p575], is an integral multiple of the allowed value step[p575], and that is less than *value* if the method invoked was the `stepDown()`[p581] method, and more than *value* otherwise.

   Otherwise (*value* subtracted from the step base[p575] is an integral multiple of the allowed value step[p575]):

   1. Let *n* be the argument.
   2. Let *delta* be the allowed value step[p575] multiplied by *n*.
   3. If the method invoked was the `stepDown()`[p581] method, negate *delta*.
   4. Let *value* be the result of adding *delta* to *value*.

8. If the element has a minimum[p574], and *value* is less than that minimum[p574], then set *value* to the smallest value that, when subtracted from the step base[p575], is an integral multiple of the allowed value step[p575], and that is more than or equal to that minimum[p574].

9. If the element has a maximum[p574], and *value* is greater than that maximum[p574], then set *value* to the largest value that, when subtracted from the step base[p575], is an integral multiple of the allowed value step[p575], and that is less than or equal to that maximum[p574].

10. If either the method invoked was the `stepDown()`[p581] method and *value* is greater than *valueBeforeStepping*, or the method invoked was the `stepUp()`[p581] method and *value* is less than *valueBeforeStepping*, then return.

    Example

    This ensures that invoking the `stepUp()`[p581] method on the `input`[p539] element in the following example does not change the value[p626] of that element:

    ```
    <input type=number value=1 max=0>
    ```

11. Let *value as string* be the result of running the algorithm to convert a number to a string[p543], as defined for the `input`[p539] element's `type`[p541] attribute's current state, on *value*.

12. Set the value[p626] of the element to *value as string*.

The **`list`** IDL attribute must return the current suggestions source element[p576], if any, or null otherwise.

The `HTMLInputElement`[p540] **`showPicker()`** and `HTMLSelectElement`[p590] **`showPicker()`** method steps are:

1. If this is not mutable[p626], then throw an `"InvalidStateError"` `DOMException`.
2. If this's relevant settings object[p1151]'s origin[p1144] is not same origin[p940] with this's relevant settings object[p1151]'s top-level origin[p1144], and this is a `select`[p590] element, or this's `type`[p541] attribute is not in the File Upload[p563] state or Color[p559] state, then throw a `"SecurityError"` `DOMException`.

   Note

   File[p563] and Color[p559] inputs are exempted from this check for historical reason: their input activation behavior[p544] also shows their pickers, and has never been guarded by an origin check.

3. If this's relevant global object[p1151] does not have transient activation[p868], then throw a `"NotAllowedError"` `DOMException`.
4. If this is a `select`[p590] element, and this is not being rendered[p1488], then throw a `"NotSupportedError"` `DOMException`.
5. Show the picker, if applicable[p582], for this.

To **show the picker, if applicable** for an `input`[p539] or `select`[p590] element *element*:

1. If *element*'s relevant global object[p1151] does not have transient activation[p868], then return.
2. If *element* is not mutable[p626], then return.

3. Consume user activation[p869] given *element*'s relevant global object[p1151].

4. If *element* does not support a picker[p544], then return.

5. If *element* is an `input`[p539] element and *element*'s `type`[p541] attribute is in the File Upload[p563] state, then run these steps in parallel[p44]:

    1. Optionally, wait until any prior execution of this algorithm has terminated.

    2. Let *dismissed* be the result of WebDriver BiDi file dialog opened with *element*.

    3. If *dismissed* is false:

        1. Display a prompt to the user requesting that the user specify some files. If the `multiple`[p572] attribute is not set on *element*, 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.

        2. Wait for the user to have made their selection.

    4. If *dismissed* is true or if the user dismissed the prompt without changing their selection, then queue an element task[p1195] on the user interaction task source[p1204] given *element* to fire an event named `cancel`[p1575] at *element*, with the `bubbles` attribute initialized to true.

    5. Otherwise, update the file selection[p563] for *element*.

    Note

    As with all user interface specifications, user agents have a good deal of freedom in how they interpret these requirements. The above text implies that a user either dismisses the prompt or changes their selection; exactly one of these will be true. But the mapping of these possibilities to specific user interface elements is not mandated by the standard. For example, a user agent might interpret clicking the "Cancel" button when files were previously selected as a change of selection to select zero files, thus firing `input` and `change`[p1575]. Or it might interpret such a click as a dismissal that leaves the selection unchanged, thus firing `cancel`[p1575]. Similarly, it's up to the user agent whether re-selecting the same files as were previously selected counts as a dismissal, or as a change of selection.

6. Otherwise, the user agent should show the relevant user interface for selecting a value for *element*, in the way it normally would when the user interacts with the control.

    When showing such a user interface, it must respect the requirements stated in the relevant parts of the specification for how *element* behaves given its `type`[p541] attribute state. (For example, various sections describe restrictions on the resulting value[p626] string.)

    This step can have side effects, such as closing other pickers that were previously shown by this algorithm. (If this closes a file selection picker, then per the above that will lead to firing either `input` and `change`[p1575] events, or a `cancel`[p1575] event.)

4.10.5.5 Common event behaviors §[p583]

When the `input` and `change`[p1575] events apply[p542] (which is the case for all `input`[p539] controls other than buttons[p532] and those with the `type`[p541] attribute in the Hidden[p546] 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`[p1575] event fires when the value is committed, if that makes sense for the control, or else when the control loses focus[p882]. In all cases, the `input` event comes before the corresponding `change`[p1575] event (if any).

When an `input`[p539] element has a defined input activation behavior[p544], the rules for dispatching these events, if they apply[p542], are given in the section above that defines the `type`[p541] attribute's state. (This is the case for all `input`[p539] controls with the `type`[p541] attribute in the Checkbox[p561] state, the Radio Button[p561] state, or the File Upload[p563] state.)

For `input`[p539] elements without a defined input activation behavior[p544], but to which these events apply[p542], and for which the user interface involves both interactive manipulation and an explicit commit action, then when the user changes the element's value[p626], the user agent must queue an element task[p1195] on the user interaction task source[p1204] given the `input`[p539] element to fire an event named `input` at the `input`[p539] 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[p1195] on the user interaction task source[p1204] given the `input`[p539] element to set its user validity[p626] to true and fire an event named `change`[p1575] at the `input`[p539] 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[p557] 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`[p1575] event would only fire when the user let go of the knob, committing to a specific value.

For `input`[p539] elements without a defined input activation behavior[p544], but to which these events apply[p542], 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[p626], the user agent must queue an element task[p1195] on the user interaction task source[p1204] given the `input`[p539] element to first fire an event named `input` at the `input`[p539] element, with the `bubbles` and `composed` attributes initialized to true, and then fire an event named `change`[p1575] at the `input`[p539] element, with the `bubbles` attribute initialized to true.

Example

An example of a user interface with a commit action would be a Color[p559] control that consists of a single button that brings up a color wheel: if the value[p626] only changes when the dialog is closed, then that would be the explicit commit action. On the other hand, if manipulating the control changes the color interactively, then there might be no commit action.

Example

Another example of a user interface with a commit action would be a Date[p550] 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`[p539] elements without a defined input activation behavior[p544], but to which these events apply[p542], any time the user causes the element's value[p626] to change without an explicit commit action, the user agent must queue an element task[p1195] on the user interaction task source[p1204] given the `input`[p539] element to fire an event named `input` at the `input`[p539] element, with the `bubbles` and `composed` attributes initialized to true. The corresponding `change`[p1575] event, if any, will be fired when the control loses focus[p882].

Example

Examples of a user changing the element's value[p626] 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.

Example

A Range[p557] control in the form of a slider that the user has focused[p875] and is interacting with using a keyboard would be another example of the user changing the element's value[p626] without a commit step.

In the case of tasks[p1194] that just fire an `input` event, user agents may wait for a suitable break in the user's interaction before queuing[p1195] 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`[p539] element's value[p626] on behalf of the user (e.g. as part of a form prefilling feature), the user agent must queue an element task[p1195] on the user interaction task source[p1204] given the `input`[p539] element to first update the value[p626] accordingly, then fire an event named `input` at the `input`[p539] element, with the `bubbles` and `composed` attributes initialized to true, then fire an event named `change`[p1575] at the `input`[p539] element, with the `bubbles` attribute initialized to true.

Note

These events are not fired in response to changes made to the values of form controls by scripts. (This is to make it easier to update the values of form controls in response to the user manipulating the controls, without having to then filter out the script's own changes to avoid an infinite loop.)

Note

These events are also not fired when the browser changes the values of form controls as part of state restoration during navigation[p1105].

4.10.6 The `button` element §[p585]

Categories[p154]:

Flow content[p157].
Phrasing content[p157].
Interactive content[p158].
Listed[p532], labelable[p532], submittable[p532], and autocapitalize-and-autocorrect inheriting[p532] form-associated element[p532].
Palpable content[p158].

Contexts in which this element can be used[p154]:

Where phrasing content[p157] is expected.
As the first child of a `select`[p590] element.

Content model[p154]:

Phrasing content[p157], but there must be no interactive content[p158] descendant and no descendant with the `tabindex`[p878] attribute specified. If the element is the first child of a `select`[p590] element, then it may also have one descendant `selectedcontent`[p624] element.

Tag omission in text/html[p154]:

Neither tag is omissible.

Content attributes[p154]:

Global attributes[p162]
`command`[p586] — Indicates to the targeted element which action to take.
`commandfor`[p586] — Targets another element to be invoked.
`disabled`[p630] — Whether the form control is disabled
`form`[p627] — Associates the element with a `form`[p532] element
`formaction`[p631] — URL to use for form submission[p657]
`formenctype`[p632] — Entry list[p661] encoding type to use for form submission[p657]
`formmethod`[p631] — Variant to use for form submission[p657]
`formnovalidate`[p632] — Bypass form control validation for form submission[p657]
`formtarget`[p632] — Navigable[p1035] for form submission[p657]
`name`[p628] — Name of the element to use for form submission[p657] and in the `form.elements`[p534] API
`popovertarget`[p935] — Targets a popover element to toggle, show, or hide
`popovertargetaction`[p935] — Indicates whether a targeted popover element is to be toggled, shown, or hidden
`type`[p586] — Type of button
`value`[p588] — Value to be used for form submission[p657]

Accessibility considerations[p154]:

For authors.
For implementers.

Sanitization[p154]:

Uncategorized[p1249] with navigating URL attributes[p1251] `formaction`[p631].

DOM interface[p154]:

IDL

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

  [CEReactions, ReflectSetter] attribute DOMString command;
  [CEReactions, Reflect="commandfor"] attribute Element? commandForElement;
  [CEReactions, Reflect] attribute boolean disabled;
  readonly attribute HTMLFormElement? form;
  [CEReactions, ReflectSetter] attribute USVString formAction;
  [CEReactions] attribute DOMString formEnctype;
  [CEReactions] attribute DOMString formMethod;
  [CEReactions, Reflect] attribute boolean formNoValidate;
  [CEReactions, Reflect] attribute DOMString formTarget;
  [CEReactions, Reflect] attribute DOMString name;
  [CEReactions, ReflectSetter] attribute DOMString type;
  [CEReactions, Reflect] 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 NodeList labels;
};
HTMLButtonElement includes PopoverTargetAttributes;
```

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 with the following keywords and states:

Keyword	State	Brief description
submit	**Submit Button**	Submits the form.
reset	**Reset Button**	Resets the form.
button	**Button**	Does nothing.

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

A button element is said to be a submit button if any of the following are true:

- the type attribute is in the Auto state, both the command and commandfor content attributes are not present, and the parent node is not a select element; or
- the type attribute is in the Submit Button state.

Constraint validation: If the element is not a submit button, the element is barred from constraint validation.

If a button element is the first child which is an element of a select element, then it is inert.

If specified, the **commandfor** attribute value must be the ID of an element in the same tree as the button with the commandfor attribute.

The **command** attribute is an enumerated attribute with the following keywords and states:

Keyword	State	Brief description
toggle-popover	**Toggle Popover**	Shows or hides the targeted popover element.
show-popover	**Show Popover**	Shows the targeted popover element.
hide-popover	**Hide Popover**	Hides the targeted popover element.
close	**Close**	Closes the targeted dialog element.
request-close	**Request Close**	Requests to close the targeted dialog element.
show-modal	**Show Modal**	Opens the targeted dialog element as modal.
A custom command keyword	**Custom**	Only dispatches the command event on the targeted element.

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

A **custom command keyword** is a string that starts with "--".

To **determine if a command is valid for a target** given a command attribute *command* and an element *target*:

1. If *command* is in the Unknown state, then return false.
2. If *command* is in the Custom state, then return true.
3. If *target* is not an HTML element, then return false.
4. If *command* is in any of the following states:

- Toggle Popover[p586]
- Show Popover[p586]
- Hide Popover[p586]

then return true.

5. If this standard does not define is valid command steps[p588] for *target*'s local name, then return false.
6. Otherwise, return the result of running *target*'s corresponding is valid command steps[p588] given *command*.

Note

The intent is to avoid dispatching a command event on an element type that does not support that command.

However, element state is not considered. If an element can support a particular command, the event is dispatched, even if that element is in a state where it cannot execute that command.

This means, for example, that command events are dispatched for popover commands on all HTML elements[p46], even if they do not have a `popover`[p925] attribute.

A `button`[p585] element *element*'s activation behavior given *event* is:

1. If *element* is disabled[p630], then return.
2. If *element*'s node document is not fully active[p1051], then return.
3. If *element* has a form owner[p626]:
    1. If *element* is a submit button[p532], then submit[p658] *element*'s form owner[p626] from *element* with *userInvolvement*[p658] set to *event*'s user navigation involvement[p1062], and return.
    2. If *element*'s `type`[p586] attribute is in the Reset Button[p586] state, then reset[p666] *element*'s form owner[p626], and return.
    3. If *element*'s `type`[p586] attribute is in the Auto[p586] state, then return.
4. Let *target* be the result of running *element*'s get the `commandfor`-associated element[p112].
5. If *target* is not null:
    1. Let *command* be *element*'s `command`[p586] attribute.
    2. If the result of determining if a command is valid for a target[p586] given *command* and *target* is false, then return.
    3. Let *continue* be the result of firing an event named `command`[p1575] at *target*, using `CommandEvent`[p872], with its `command`[p873] attribute initialized to *command*, its `source`[p873] attribute initialized to *element*, and its `cancelable` attribute initialized to true.

        DOM standard issue #1328 tracks how to better standardize associated event data in a way which makes sense on Events. Currently an event attribute initialized to a value cannot also have a getter, and so an internal slot (or map of additional fields) is required to properly specify this.

    4. If *continue* is false, then return.
    5. If *target* is not connected, then return.
    6. If *command* is in the Custom[p586] state, then return.
    7. If *command* is in the Hide Popover[p586] state:
        1. Let *validity* be the result of running check popover validity[p934] given *target*, true, and null. If this throws an exception, catch it, and let *validity* be false.
        2. If *validity* is true, then run the hide popover algorithm[p930] given *target*, true, true, false, and *element*.
    8. Otherwise, if *command* is in the Toggle Popover[p586] state:
        1. Let *validity* be the result of running check popover validity[p934] given *target*, false, and null. If this throws

an exception, catch it, and let *validity* be false.

2. If *validity* is true, then run the show popover algorithm[p927] given *target*, false, and *element*.

3. Otherwise:

    1. Let *validity* be the result of running check popover validity[p934] given *target*, true, and null. If this throws an exception, catch it, and let *validity* be false.

    2. If *validity* is true, then run the hide popover algorithm[p930] given *target*, true, true, false, and *element*.

9. Otherwise, if *command* is in the Show Popover[p586] state:

    1. Let *validity* be the result of running check popover validity[p934] given *target*, false, and null. If this throws an exception, catch it, and let *validity* be false.

    2. If *validity* is true, then run the show popover algorithm[p927] given *target*, false, and *element*.

10. Otherwise, if this standard defines command steps[p588] for *target*'s local name, then run the corresponding command steps[p588] given *target*, *element*, and *command*.

6. Otherwise, run the popover target attribute activation behavior[p936] given *element* and *event*'s target.

An HTML element[p46] can have specific **is valid command steps** and **command steps** defined for the element's local name.

The `form`[p627] attribute is used to explicitly associate the `button`[p585] element with its form owner[p626]. The `name`[p628] attribute represents the element's name. The `disabled`[p630] attribute is used to make the control non-interactive and to prevent its value from being submitted. The `formaction`[p631], `formenctype`[p632], `formmethod`[p631], `formnovalidate`[p632], and `formtarget`[p632] attributes are attributes for form submission[p631].

Note

The `formnovalidate`[p632] attribute can be used to make submit buttons that do not trigger the constraint validation.

The `formaction`[p631], `formenctype`[p632], `formmethod`[p631], `formnovalidate`[p632], and `formtarget`[p632] must not be specified if the element is not a submit button[p532].

The **`command`** getter steps are:

1. Let *command* be this's `command`[p586] attribute.

2. If *command* is in the Custom[p586] state, then return *command*'s value.

3. If *command* is in the Unknown[p586] state, then return the empty string.

4. Return the keyword corresponding to the value of *command*.

The **`value`** attribute gives the element's value for the purposes of form submission. The element's value[p626] is the value of the element's `value`[p588] attribute, if there is one; otherwise the empty string. The element's optional value[p626] is the value of the element's `value`[p588] attribute, if there is one; otherwise null.

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 **`type`** getter steps are:

1. If this is a submit button[p532], then return "`submit`".

2. Let *state* be this's `type`[p586] attribute.

3. Assert: *state* is not in the Submit Button[p586] state.

4. If *state* is in the Auto[p586] state, then return "`button`".

5. Return the keyword value corresponding to *state*.

The willValidate[p654], validity[p654], and validationMessage[p656] IDL attributes, and the checkValidity()[p656], reportValidity()[p656], and setCustomValidity()[p654] methods, are part of the constraint validation API[p653]. The labels[p538] IDL attribute provides a list of the element's label[p536]s. The disabled[p585], form[p628], and name[p585] 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:

```
<button type=button
        onclick="alert('This 15-20 minute piece was composed by George Gershwin.')">
 Show hint
</button>
```

Example

The following shows how buttons[p532] can use commandfor[p586] to show and hide an element with the popover[p925] attribute when activated:

```
<button type=button
        commandfor="the-popover"
        command="show-popover">
 Show menu
</button>
<div popover
     id="the-popover">
 <button commandfor="the-popover"
         command="hide-popover">
  Hide menu
 </button>
</div>
```

Example

The following shows how buttons can use commandfor[p586] with a custom command keyword[p586] on an element, demonstrating how one could use custom commands for unspecified behavior:

```
<button type=button
        commandfor="the-image"
        command="--rotate-landscape">
 Rotate Left
</button>
<button type=button
        commandfor="the-image"
        command="--rotate-portrait">
 Rotate Right
</button>
<img id="the-image"
     src="photo.jpg">
<script>
  const image = document.getElementById("the-image");
  image.addEventListener("command", (event) => {
   if ( event.command == "--rotate-landscape" ) {
    event.target.style.rotate = "-90deg"
   } else if ( event.command == "--rotate-portrait" ) {
    event.target.style.rotate = "0deg"
   }
  });
</script>
```

4.10.7 The select element §[p590]

Categories[p154]:

Flow content[p157].
Phrasing content[p157].
Interactive content[p158].
Listed[p532], labelable[p532], submittable[p532], resettable[p532], and autocapitalize-and-autocorrect inheriting[p532] form-associated element[p532].
Palpable content[p158].

Contexts in which this element can be used[p154]:

Where phrasing content[p157] is expected.

Content model[p154]:

Zero or one button[p585] elements if the select[p590] is a drop-down box[p1517], followed by zero or more option[p601] elements, optgroup[p600] elements, hr[p241] elements, script-supporting elements[p159], noscript[p703] elements, or div[p266] elements.

Tag omission in text/html[p154]:

Neither tag is omissible.

Content attributes[p154]:

Global attributes[p162]
autocomplete[p633] — Hint for form autofill feature
disabled[p630] — Whether the form control is disabled
form[p627] — Associates the element with a form[p532] element
multiple[p591] — Whether to allow multiple values
name[p628] — Name of the element to use for form submission[p657] and in the form.elements[p534] API
required[p591] — Whether the control is required for form submission[p657]
size[p591] — Size of the control

Accessibility considerations[p154]:

If the element has a multiple[p591] attribute or a size[p591] attribute with a value > 1: for authors; for implementers.
Otherwise: for authors; for implementers.

Sanitization[p154]:

Uncategorized[p1249].

DOM interface[p154]:

IDL

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

  [CEReactions, ReflectSetter] attribute DOMString autocomplete;
  [CEReactions, Reflect] attribute boolean disabled;
  readonly attribute HTMLFormElement? form;
  [CEReactions, Reflect] attribute boolean multiple;
  [CEReactions, Reflect] attribute DOMString name;
  [CEReactions, Reflect] attribute boolean required;
  [CEReactions, Reflect, ReflectDefault=0] attribute unsigned long size;


  readonly attribute DOMString type;

  [SameObject] readonly attribute HTMLOptionsCollection options;
  [CEReactions] attribute unsigned long length;
  getter HTMLOptionElement? 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;
  [CEReactions] attribute long selectedIndex;
  [CEReactions] attribute DOMString value;

  readonly attribute boolean willValidate;
  readonly attribute ValidityState validity;
  readonly attribute DOMString validationMessage;
  boolean checkValidity();
  boolean reportValidity();
  undefined setCustomValidity(DOMString error);

  undefined showPicker();

  readonly attribute NodeList labels;
};
```

The select element represents a control for selecting amongst a set of options.

The **multiple** attribute is a boolean attribute. If the attribute is present, then the select element represents a control for selecting zero or more options from the list of options. If the attribute is absent, then the select element represents a control for selecting a single option from the list of options.

The **size** attribute gives the number of options to show to the user. The size attribute, if specified, must have a value that is a valid non-negative integer greater than zero.

The **required** attribute is a boolean attribute. When specified, the user will be required to select a value before submitting the form.

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.

If a select element has a required 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 does not have an ancestor optgroup in between itself and the select 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.

The reset algorithm for a select element *selectElement* is:

1. Set *selectElement*'s user validity to false.
2. For each *optionElement* of *selectElement*'s list of options:
    1. If *optionElement* has a selected attribute, then set *optionElement*'s selectedness to true; otherwise set it to false.
    2. Set *optionElement*'s dirtiness to false.
3. Run the selectedness setting algorithm given *selectElement*.
4. Update a select's descendant selectedcontent elements given *selectElement*.

A select element that is not disabled is mutable.

The user agent should allow the user to pick an option element from a select element in its list of options (either through a click, or through unfocusing the element after changing its value, or through a menu command, or through any other mechanism) by running the pick an option algorithm given the select element, the option element, and if the select's contents are being rendered with base appearance, a corresponding keydown or mouseup event, otherwise null.

To **pick an option** given a select element *select*, an option element *option*, and an Event or null *event*:

1. If *select* has the multiple attribute or *select* is disabled, then return.
2. If *event* is not null and *event*'s canceled flag is set, then return.
3. If *event* is a MouseEvent and *event*'s button attribute is not 0, then return.
4. Set *option*'s selectedness to true.
5. Set *option*'s dirtiness to true.
6. Send select update notifications given *select*.
7. If *select* is being rendered as a drop-down box with base appearance, then run the hide popover algorithm given *select*'s select popover.

If the multiple attribute is absent, whenever an option element in the select element's list of options has its selectedness set to true, and whenever an option element with its selectedness set to true is added to the select element's list of options, the user agent must set the selectedness of all the other option elements in its list of options to false.

If the multiple attribute is absent and the element's display size is greater than 1, then the user agent should also allow the user to request that the option whose selectedness is true, if any, be unselected. Upon this request being conveyed to the user agent, and before the relevant user interaction event is queued (e.g. before the click event), the user agent must set the selectedness of that option element to false, set its dirtiness to true, and then send select update notifications.

If the select is being rendered as a drop-down box with base appearance, then the user agent should allow the user to **open the picker** given a corresponding select element *select* and a corresponding mousedown or keydown event *event*:

1. If *event*'s canceled flag is set, then return.
2. If *event* is a MouseEvent and *event*'s button attribute is not 0, then return.
3. Run the show popover algorithm given *select*'s select popover, false, and *select*.

If the select is being rendered with base appearance, then the user agent should allow the user to **focus another option** given the new option element to focus *option* and a keydown event *event*:

1. If *event*'s canceled flag is set, then return.
2. Run the focusing steps on *option*.

Note

Implementations commonly allow the user to focus the next or previous option via the arrow-up and arrow-down keys, focus the first or last option via the Home or End keys, or focus the first or last option in the viewport of the picker via the PageUp or PageDown keys.

Note

Which particular keys of the keyboard cause these actions might differ across implementations and platforms. The ARIA Authoring Practices Guide has good recommendations for this behavior.

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[p603] of the element must be set to true, and the user agent must send select update notifications[p594].

The **display size** of a select[p590] element is the result of applying the rules for parsing non-negative integers[p81] to the value of the element's size[p591] 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[p591] attribute is absent, then the element's display size[p593] is 4 if the element's multiple[p591] content attribute is present, and 1 otherwise.

To get the **list of options** given a select[p590] element *select*:

1. Let *options* be « ».
2. Let *node* be the first child of *select* in tree order.
3. While *node* is not null:
    1. If *node* is an option[p601] element, then append *node* to *options*.
    2. If any of the following conditions are true:
        - *node* is a select[p590] element;
        - *node* is an hr[p241] element;
        - *node* is an option[p601] element;
        - *node* is a datalist[p598] element;
        - *node* is an optgroup[p600] element and *node* has an ancestor optgroup[p600] in between itself and *select*,

        then set *node* to the next descendant of *select* in tree order, excluding *node*'s descendants, if any such node exists; otherwise null.

        Otherwise, set *node* to the next descendant of *select* in tree order, if any such node exists; otherwise null.
4. Return *options*.

To **ask for a reset** given an option[p601] element *option*:

1. Let *select* be *option*'s cached nearest ancestor select element[p604].
2. If *select* is null, then return.
3. Run the selectedness setting algorithm[p593] given *select*.
4. Update a select's descendant selectedcontent elements[p625] given *select*.

To run the **selectedness setting algorithm** given a select[p590] element *select*, run these steps. They return a boolean.

1. If *select* has the multiple[p591] attribute, then return false.
2. Let *hasChanged* be false.
3. Let *firstEnabledOption* be null.
4. Let *lastSelectedOption* be null.
5. For each *option* of *select*'s list of options[p593]:
    1. If *option*'s selectedness[p603] is true:
        1. If *lastSelectedOption* is not null:
            1. Set *lastSelectedOption*'s selectedness[p603] to false.
            2. Set *hasChanged* to true.
        2. Set *lastSelectedOption* to *option*.
    2. If *firstEnabledOption* is null and *option* is not disabled[p602], then set *firstEnabledOption* to *option*.

6. If *lastSelectedOption* is null, *firstEnabledOption* is not null, and *select*'s display size[p593] is 1:

    1. Set *firstEnabledOption*'s selectedness[p603] to true.

    2. Set *hasChanged* to true.

7. Return *hasChanged*.

To **reset a `select`'s selectedness**, given a `select`[p590] element *select* and a boolean *hasChanged*:

1. If the result of running the selectedness setting algorithm[p593] given *select* is true, then set *hasChanged* to true.

2. If *hasChanged* is false, then return.

3. Update a `select`'s descendant `selectedcontent` elements[p625] given *select*.

Whenever a `select`[p590] element's `size`[p591] attribute is set, changed, or removed, reset a `select`'s selectedness[p594] given the `select`[p590] element and false.

Whenever a `select`[p590] element *select*'s `multiple`[p591] attribute is added or removed, run these steps:

1. If the `multiple`[p591] attribute was removed, then set the selectedness[p603] of all but the first `option`[p601] element with its selectedness[p603] set to true in *select*'s list of options[p593] to false.

2. Reset a `select`'s selectedness[p594] given *select* and true.

Note

The `selectedcontent`[p624] elements of a `select`[p590] element are not kept up to date while it has a `multiple`[p591] attribute, which is why they are always updated here.

To **send `select` update notifications** for a `select`[p590] element *element*, queue an element task[p1195] on the user interaction task source[p1204] given *element* to run these steps:

1. Set *element*'s user validity[p626] to true.

2. Update a `select`'s descendant `selectedcontent` elements[p625] given *element*.

3. Clone selected `option` into `select` button[p605] given *element*.

4. Fire an event named `input` at *element*, with the `bubbles` and `composed` attributes initialized to true.

5. Fire an event named `change`[p1575] at *element*, with the `bubbles` attribute initialized to true.

For web developers (non-normative)

***select*.type**[p595]

Returns "`select-multiple`" if the element has a `multiple`[p591] attribute, and "`select-one`" otherwise.

***select*.options**[p595]

Returns an `HTMLOptionsCollection`[p119] of the list of options[p593].

***select*.length**[p595] **[= *value*]**

Returns the number of elements in the list of options[p593].

When set to a smaller number, truncates the number of `option`[p601] elements in the `select`[p590].

When set to a greater number, adds new blank `option`[p601] elements to the `select`[p590].

***element* = *select*.item**[p595]**(*index*)**

***select*[*index*]**

Returns the item with index *index* from the list of options[p593]. The items are sorted in tree order.

***element* = *select*.namedItem**[p595]**(*name*)**

Returns the first item with ID or `name`[p1531] *name* from the list of options[p593].

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 [= *value*]**

Returns the index of the first selected item, if any, or −1 if there is no selected item.

Can be set, to change the selection.

***select*.value [= *value*]**

Returns the value of the first selected item, if any, or the empty string if there is no selected item.

Can be set, to change the selection.

***select*.showPicker()**

Shows any applicable picker UI for *select*, so that the user can select a value.

Throws an "InvalidStateError" DOMException if *select* is not mutable.

Throws a "NotAllowedError" DOMException if called without transient user activation.

Throws a "SecurityError" DOMException if *select* is inside a cross-origin iframe.

Throws a "NotSupportedError" DOMException if *select* is not being rendered.

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(*element*, *before*)** 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.

A select element's **selected index** is the index of the first option element in its list of options in tree order that has its selectedness set to true. If there isn't one, then it is −1.

To **set the selected index** of a select element *select* to a value *value*:

1. Let *firstMatchingOption* be null.
2. For each *option* of *select*'s list of options[p593]:
    1. Set *option*'s selectedness[p603] to false.
    2. If *firstMatchingOption* is null and *option*'s index[p604] is equal to *value*, then set *firstMatchingOption* to *option*.
3. If *firstMatchingOption* is not null, then set *firstMatchingOption*'s selectedness[p603] to true and set *firstMatchingOption*'s dirtiness[p603] to true.
4. Update a `select`'s descendant `selectedcontent` elements[p625] given *select*.

Note

This can result in no element having a selectedness[p603] set to true even in the case of the `select`[p590] element having no `multiple`[p591] attribute and a display size[p593] of 1.

The **`selectedIndex`** getter steps are to return this's selected index[p595].

The `selectedIndex`[p596] setter steps are to set the selected index[p595] of this to the given value.

The **`value`** getter steps are to return the value[p603] of the first `option`[p601] element in this's list of options[p593] in tree order that has its selectedness[p603] set to true, if any. If there isn't one, then return the empty string.

The `value`[p596] setter steps are:

1. Let *firstMatchingOption* be null.
2. For each *option* of this's list of options[p593]:
    1. Set *option*'s selectedness[p603] to false.
    2. If *firstMatchingOption* is null and *option*'s value[p603] is equal to the given value, then set *firstMatchingOption* to *option*.
3. If *firstMatchingOption* is not null, then set *firstMatchingOption*'s selectedness[p603] to true and set *firstMatchingOption*'s dirtiness[p603] to true.
4. Update a `select`'s descendant `selectedcontent` elements[p625] given this.

Note

This can result in no element having a selectedness[p603] set to true even in the case of the `select`[p590] element having no `multiple`[p591] attribute and a display size[p593] of 1.

Note

For historical reasons, the default value of the `size`[p590] IDL attribute does not return the actual size used, which, in the absence of the `size`[p591] content attribute, is either 1 or 4 depending on the presence of the `multiple`[p591] attribute.

The `willValidate`[p654], `validity`[p654], and `validationMessage`[p656] IDL attributes, and the `checkValidity()`[p656], `reportValidity()`[p656], and `setCustomValidity()`[p654] methods, are part of the constraint validation API[p653]. The `labels`[p538] IDL attribute provides a list of the element's `label`[p536]s. The `disabled`[p590], `form`[p628], and `name`[p590] IDL attributes are part of the element's forms API.

Example

The following example shows how a `select`[p590] element can be used to offer the user with a set of options from which the user can select a single option. The default option is preselected.

```
<p>
 <label for="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:

```
<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.

```
<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.

```
<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 value="s3">The More You Ruv Someone
  <option value="s4">Schadenfreude
  <option value="s5">I Wish I Could Go Back to College
  <option value="s6">The Money Song
  <option value="s7">School for Monsters
  <option value="s8">The Money Song (Reprise)
  <option value="s9">There's a Fine, Fine Line (Reprise)
  <option value="s10">What Do You Do With a B.A. in English? (Reprise)
  <option value="s11">For Now
 </select>
</label>
```

Example

Occasionally it can be useful to have a separator:

```
<label>
 Select the song to play next:
```

```
<select required name="next">
 <option value="sr">Random
 <hr>
 <option value="s1">It Sucks to Be Me (Reprise)
 <option value="s2">There is Life Outside Your Apartment
 …
```

¶ p598 Example

Here is an example which uses div p266, legend p623, img p359, button p585, and selectedcontent p624 elements:

```
<select>
  <button>
    <selectedcontent></selectedcontent>
  </button>
  <div class="border">
    <optgroup>
      <legend>WHATWG Specifications</legend>
      <option>
        <img src="html.jpg" alt="">
        HTML
      </option>
      <option>
        <img src="dom.jpg" alt="">
        DOM
      </option>
    </optgroup>
  </div>
  <div class="border">
    <optgroup>
      <legend>W3C Specifications</legend>
      <option>
        <img src="forms.jpg" alt="">
        CSS Form Control Styling
      </option>
      <option>
        <img src="pseudo.jpg" alt="">
        CSS Pseudo-Elements
      </option>
    <optgroup>
  </div>
</select>
```

¶ p598 Note

The first child button p585 element as allowed by the content model of select p590 is not a submit button. It is used to replace the in-page rendering of the select p590 element. Its form submission behavior is prevented because it is inert p866. A selectedcontent p624 element can be used inside this button p585 element to display the contents of the currently selected option p601 element.

4.10.8 The datalist element § p598

Categories p154:

Flow content p157.
Phrasing content p157.

Contexts in which this element can be used p154:

Where phrasing content p157 is expected.

Content model[p154]**:**

Either: phrasing content[p157].
Or: Zero or more option[p601] and script-supporting[p159] elements.

Tag omission in text/html[p154]**:**

Neither tag is omissible.

Content attributes[p154]**:**

Global attributes[p162]

Accessibility considerations[p154]**:**

For authors.
For implementers.

Sanitization[p154]**:**

Uncategorized[p1249].

DOM interface[p154]**:**

IDL

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

  [SameObject] readonly attribute HTMLCollection options;
};
```

The datalist[p598] element represents a set of option[p601] elements that represent predefined options for other controls. In the rendering, the datalist[p598] element represents[p148] nothing and it, along with its children, should be hidden.

The datalist[p598] element can be used in two ways. In the simplest case, the datalist[p598] element has just option[p601] element children.

Example

```
<label>
 Animal:
 <input name=animal list=animals>
 <datalist id=animals>
  <option value="Cat">
  <option value="Dog">
 </datalist>
</label>
```

In the more elaborate case, the datalist[p598] element can be given contents that are to be displayed for down-level clients that don't support datalist[p598]. In this case, the option[p601] elements are provided inside a select[p590] element inside the datalist[p598] element.

Example

```
<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)

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 descendant of a select element.

Content model:

Zero or one legend element followed by zero or more option elements, script-supporting elements, noscript elements, or div 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, if it is immediately followed by an hr 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.

Sanitization:

Uncategorized.

DOM interface:

IDL

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

  [CEReactions, Reflect] attribute boolean disabled;
  [CEReactions, Reflect] 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 descendants 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.

Whenever an optgroup element's disabled attribute is added or removed, then:

1. Let *select* be the optgroup element's nearest ancestor select.
2. If *select* is null, then return.
3. Reset a select's selectedness given *select* and false.

The **label** attribute must be specified if the optgroup has no child legend element.

To **get an optgroup element's label**, given an optgroup *optgroup*:

1. If *optgroup*'s first element child is a legend element, then return the result of get HTML-aware text content given that legend element and true.
2. If *optgroup* has a label attribute, then return the value of that attribute.
3. Return the empty string.

The value of the optgroup label algorithm 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.

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:

```
<form action="courseselector.dll" method="get">
 <p>Which course would you like to watch today?
 <p><label>Course:
  <select name="c">
   <optgroup label="8.01 Physics I: Classical Mechanics">
    <option value="8.01.1">Lecture 01: Powers of Ten
    <option value="8.01.2">Lecture 02: 1D Kinematics
    <option value="8.01.3">Lecture 03: Vectors
   <optgroup label="8.02 Electricity and Magnetism">
    <option value="8.02.1">Lecture 01: What holds our world together?
    <option value="8.02.2">Lecture 02: Electric Field
    <option value="8.02.3">Lecture 03: Electric Flux
   <optgroup label="8.03 Physics III: Vibrations and Waves">
    <option value="8.03.1">Lecture 01: Periodic Phenomenon
    <option value="8.03.2">Lecture 02: Beats
    <option value="8.03.3">Lecture 03: Forced Oscillations with Damping
  </select>
 </label>
 <p><input type=submit value="▶ Play">
</form>
```

4.10.10 The option element §

Categories:

None.

Contexts in which this element can be used:

As a descendant of a select element.
As a descendant of a datalist element.
As an optgroup element.

Content model:

If the element has a label attribute and a value attribute: Nothing.

If the element has a label[p603] attribute but no value[p603] attribute: Text[p158].
If the element has no label[p603] attribute and is not a descendant of a datalist[p598] element: Zero or more div[p266] elements or phrasing content[p157], but there must be no interactive content[p158] descendant, datalist[p598] element descendant, object[p416] element descendant, or descendant with the tabindex[p878] attribute specified
If the element has no label[p603] attribute and is a descendant of a datalist[p598] element: Text[p158].

Tag omission in text/html[p154]:

An option[p601] element's end tag[p1359] can be omitted if the option[p601] element is immediately followed by another option[p601] element, if it is immediately followed by an optgroup[p600] element, if it is immediately followed by an hr[p241] element, or if there is no more content in the parent element.

Content attributes[p154]:

Global attributes[p162]
disabled[p602] — Whether the form control is disabled
label[p603] — User-visible label
selected[p603] — Whether the option is selected by default
value[p603] — Value to be used for form submission[p657]

Accessibility considerations[p154]:

For authors.
For implementers.

Sanitization[p154]:

Uncategorized[p1249].

DOM interface[p154]:

IDL

```
[Exposed=Window,
 LegacyFactoryFunction=Option(optional DOMString text = "", optional DOMString value, optional
boolean defaultSelected = false, optional boolean selected = false)]
interface HTMLOptionElement : HTMLElement {
  [HTMLConstructor] constructor();

  [CEReactions, Reflect] attribute boolean disabled;
  readonly attribute HTMLFormElement? form;
  [CEReactions, ReflectSetter] attribute DOMString label;
  [CEReactions, Reflect="selected"] attribute boolean defaultSelected;
  [CEReactions] attribute boolean selected;
  [CEReactions, ReflectSetter] attribute DOMString value;

  [CEReactions] attribute DOMString text;
  readonly attribute long index;
};
```

The option[p601] element represents[p148] an option in a select[p590] element or as part of a list of suggestions in a datalist[p598] element.

In certain circumstances described in the definition of the select[p590] element, an option[p601] element can be a select[p590] element's placeholder label option[p591]. A placeholder label option[p591] does not represent an actual option, but instead represents a label for the select[p590] control.

The **disabled** attribute is a boolean attribute[p79]. An option[p601] element *option* is **disabled** if the following steps return true:

1. If *option*'s disabled[p602] attribute is present, then return true.
2. For each *ancestor* of *option*'s ancestors in reverse tree order:
    1. If *ancestor* is a select[p590], hr[p241], datalist[p598], or option[p601] element, then return false.
    2. If *ancestor* is an optgroup[p600] element, then return true if *ancestor*'s disabled[p600] attribute is present; otherwise false.
3. Return false.

Whenever an option[p601] element's disabled[p602] attribute is added or removed, then:

1. Let *select* be the option element's cached nearest ancestor select element.
2. If *select* is null, then return.
3. Reset a select's selectedness given *select* and false.

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.

Note

Being disabled does not prevent all modifications to the option element. For example, its selectedness could be modified programmatically from JavaScript. Or, it could be indirectly modified by user action, e.g., if other non-disabled option elements in the select element were modified.

The **label** attribute provides a label for the element.

To **get an option element's label**, given an option element *option* and a boolean *includeAltText*:

1. If *option* has a label attribute whose value is not the empty string, then return that value.
2. Return the result of get HTML-aware text content given *option* and *includeAltText*.

The label content attribute, if specified, must not be empty.

The **value** attribute provides a value for the 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 result of get HTML-aware text content given this and false.

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 *option*'s selected attribute is added or removed:

1. Let *hasChanged* be false.
2. If *option*'s dirtiness is false:
    1. If the selected attribute was added and *option*'s selectedness is false, then set *option*'s selectedness to true and set *hasChanged* to true.
    2. If the selected attribute was removed and *option*'s selectedness is true, then set *option*'s selectedness to false and set *hasChanged* to true.
3. Let *select* be *option*'s cached nearest ancestor select element.
4. If *select* is null, then return.
5. Reset a select's selectedness given *select* and *hasChanged*.

Note

For example, in a select element without a multiple attribute, adding the selected attribute to an option element that is not selected deselects the option element that was. The selectedness setting algorithm then has nothing left to change, so without hasChanged *the selectedcontent elements would not be updated.*

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[p603] attribute set.

An option[p601] element's **index** is the number of option[p601] elements that are in the same list of options[p593] but that come before it in tree order. If the option[p601] element is not in a list of options[p593], then the option[p601] element's index[p604] is zero.

Each option[p601] element has a **cached nearest ancestor select element**, which is a select[p590] element or null, initially set to null.

To **update an option's nearest ancestor select**, given an option[p601] *option*, run these steps. They return a list of select[p590] elements whose descendant selectedcontent[p624] elements need updating.

1. Let *oldSelect* be *option*'s cached nearest ancestor select element[p604].
2. Let *newSelect* be *option*'s nearest ancestor select[p604].
3. Set *option*'s cached nearest ancestor select element[p604] to *newSelect*.
4. If *oldSelect* is *newSelect*, then return « ».
5. Let *result* be « ».
6. For each *select* of « *oldSelect*, *newSelect* »:
    1. If *select* is null, then continue.
    2. Let *hasChanged* be the result of running the selectedness setting algorithm[p593] given *select*.
    3. If *option*'s selectedness[p603] is true or *hasChanged* is true, then append *select* to *result*.
7. Return *result*.

The option[p601] HTML element insertion steps[p46], given *insertedOption*, are to update an option's nearest ancestor select[p604] given *insertedOption*.

Note

The returned select[p590] elements are intentionally ignored here. Any selectedcontent[p624] update is instead performed by the option[p601] HTML element post-connection steps[p46], which run once the tree is in a consistent state.

The option[p601] HTML element post-connection steps[p46], given *insertedOption*, are to update descendant selectedcontent elements for an option[p604] given *insertedOption*.

To **update descendant selectedcontent elements for an option**, given an option[p601] element *option*:

1. Let *select* be *option*'s cached nearest ancestor select element[p604].
2. If *select* is null or *option*'s selectedness[p603] is false, then return.
3. Update a select's descendant selectedcontent elements[p625] given *select*.

The option[p601] HTML element removing steps[p46], given *removedNode*, *isSubtreeRoot*, and *oldAncestor*, are:

1. Let *selects* be the result of update an option's nearest ancestor select[p604] given *removedNode*.
2. For each *select* of *selects*: queue a microtask[p1195] to update a select's descendant selectedcontent elements[p625] given *select*.

The option[p601] HTML element moving steps[p46], given *movedNode*, *isSubtreeRoot*, and *oldAncestor*, are:

1. Let *selects* be the result of update an option's nearest ancestor select[p604] given *movedNode*.
2. For each *select* of *selects*: queue a microtask[p1195] to update a select's descendant selectedcontent elements[p625] given *select*.

To get the **nearest ancestor select** given an Element *element*, run these steps. They return a select[p590] element or null.

1. Let *ancestorOptgroup* be null.
2. For each *ancestor* of *element*'s ancestors, in reverse tree order:

1. If *ancestor* is a datalist[p598], hr[p241], or option[p601] element, then return null.
2. If *ancestor* is an optgroup[p600] element:
    1. If *ancestorOptgroup* is not null, then return null.
    2. Set *ancestorOptgroup* to *ancestor*.
3. If *ancestor* is a select[p590], then return *ancestor*.

3. Return null.

To **clone selected `option` into `select` button**, given a select[p590] element *select*:

1. Let *option* be the first element of *select*'s option list[p593] whose selectedness[p603] is set to true, if such an element exists; otherwise null.
2. Let *text* be the empty string.
3. If *option* is not null, then set *text* to *option*'s label[p605].
4. Set *select*'s select fallback button text[p1518] to *text*.

When an option[p601] element *option* is popped off the stack of open elements[p1383] of an HTML parser[p1368] or XML parser[p1484], update descendant selectedcontent elements for an option[p604] given *option*.

For web developers (non-normative)

***option*.selected**[p605]

Returns true if the element is selected, and false otherwise.

Can be set, to override the current state of the element.

***option*.index**[p606]

Returns the index of the element in its select[p590] element's options[p595] list.

***option*.form**[p606]

Returns the element's form[p532] element, if any, or null otherwise.

***option*.text**[p606]

Same as textContent, except that spaces are collapsed and script[p685] elements are skipped.

***option* = new Option**[p606]**([*text* [, *value* [, *defaultSelected* [, *selected*]]]])**

Returns a new option[p601] element.

The *text* argument sets the contents of the element.

The *value* argument sets the value[p603] attribute.

The *defaultSelected* argument sets the selected[p603] 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 **`label`** getter steps are:

1. Let *attribute* be this's label[p603] attribute.
2. If *attribute* is null, then return the result of getting an option element's label[p603] given this and false.
3. Return *attribute*'s value.

The **`value`** getter steps are to return this's value[p603].

The **`selected`** getter steps are to return true if this's selectedness[p603] is true, and false otherwise.

The selected[p605] setter steps are:

1. Set this's selectedness[p603] to the given value.
2. Set this's dirtiness[p603] to true.

3. Ask for a reset[p593] given this.

The **index** IDL attribute must return the element's index[p604].

The **text** getter steps are to return the result of get HTML-aware text content[p606] given this and false.

The text[p606] setter steps are to string replace all with the given value within this.

The **form** getter steps are:

1. Let *select* be this's nearest ancestor select[p604].
2. If *select* is null, then return null.
3. Return *select*'s form owner[p626].

A legacy factory function is provided for creating HTMLOptionElement[p602] 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[p1151]'s associated Document[p966].
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[p603]" and *value*.
5. If *defaultSelected* is true, then set an attribute value for *option* using "selected[p603]" and the empty string.
6. If *selected* is true, then set *option*'s selectedness[p603] to true; otherwise set its selectedness[p603] to false (even if *defaultSelected* is true).
7. Return *option*.

To **get HTML-aware text content**, given an element *element* and a boolean *includeAltText*:

1. Let *text* be the empty string.
2. For each descendant *descendant* of *element* in tree order:
    1. If *descendant* is a script[p685] or SVG script element, then continue skipping all descendants of *descendant*.
    2. If *descendant* is a Text node, then set *text* to the concatenation of *text* and *descendant*'s data.
    3. If *descendant* is an img[p359] element with an alt[p360] attribute and *includeAltText* is true, then set *text* to the concatenation of *text* and the value of *descendant*'s alt[p360] attribute.
    4. If *descendant* is an img[p359] element, then continue skipping all descendants of *descendant*.
3. Return the result of strip and collapse ASCII whitespace given *text*.

¶[p606] Note

When no value[p603] attribute is set on the option[p601] element, its text contents are used to generate a submittable value. In the case that the option[p601] element has child elements, this can lead to unexpected results such as option[p601] elements which render differently but have the same value. In order to address this, setting the value[p603] attribute on option[p601] elements is recommended.

4.10.11 The textarea element §[p606]

Categories[p154]:

Flow content[p157].
Phrasing content[p157].
Interactive content[p158].
Listed[p532], labelable[p532], submittable[p532], resettable[p532], and autocapitalize-and-autocorrect inheriting[p532] form-associated

element[p532].
Palpable content[p158].

Contexts in which this element can be used[p154]:

Where phrasing content[p157] is expected.

Content model[p154]:

Text[p158].

Tag omission in text/html[p154]:

Neither tag is omissible.

Content attributes[p154]:

Global attributes[p162]
autocomplete[p633] — Hint for form autofill feature
cols[p609] — Maximum number of characters per line
dirname[p629] — Name of form control to use for sending the element's directionality[p168] in form submission[p657]
disabled[p630] — Whether the form control is disabled
form[p627] — Associates the element with a form[p532] element
maxlength[p609] — Maximum length of value
minlength[p609] — Minimum length of value
name[p628] — Name of the element to use for form submission[p657] and in the form.elements[p534] API
placeholder[p610] — User-visible label to be placed within the form control
readonly[p608] — Whether to allow the value to be edited by the user
required[p610] — Whether the control is required for form submission[p657]
rows[p609] — Number of lines to show
wrap[p609] — How the value of the form control is to be wrapped for form submission[p657]

Accessibility considerations[p154]:

For authors.
For implementers.

Sanitization[p154]:

Uncategorized[p1249].

DOM interface[p154]:

IDL

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

  [CEReactions, ReflectSetter] attribute DOMString autocomplete;
  [CEReactions, ReflectPositiveWithFallback, ReflectDefault=20] attribute unsigned long cols;
  [CEReactions, Reflect] attribute DOMString dirName;
  [CEReactions, Reflect] attribute boolean disabled;
  readonly attribute HTMLFormElement? form;
  [CEReactions, ReflectNonNegative] attribute long maxLength;
  [CEReactions, ReflectNonNegative] attribute long minLength;
  [CEReactions, Reflect] attribute DOMString name;
  [CEReactions, Reflect] attribute DOMString placeholder;
  [CEReactions, Reflect] attribute boolean readOnly;
  [CEReactions, Reflect] attribute boolean required;
  [CEReactions, ReflectPositiveWithFallback, ReflectDefault=2] attribute unsigned long rows;
  [CEReactions, Reflect] 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[p606] element represents[p148] 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[p608] of a textarea[p606] control must be initially the empty string.

Note

This element has rendering requirements involving the bidirectional algorithm[p177].

The **readonly** attribute is a boolean attribute[p79] 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:

```
Filename: <code>/etc/bash.bashrc</code>
<textarea name="buffer" readonly>
# System-wide .bashrc file for interactive bash(1) shells.

# To enable the settings / commands in this file for login shells as well,
# this file has to be sourced in /etc/profile.

# If not running interactively, don't do anything
[ -z "$PS1" ] &amp;&amp; return

...</textarea>
```

Constraint validation: If the readonly[p608] attribute is specified on a textarea[p606] element, the element is barred from constraint validation[p651].

A textarea[p606] element is mutable[p626] if it is neither disabled[p630] nor has a readonly[p608] attribute specified.

When a textarea[p606] is mutable[p626], its raw value[p608] 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[p608] to change, the user agent must queue an element task[p1195] on the user interaction task source[p1204] given the textarea[p606] element to fire an event named input at the textarea[p606] 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[p606] element's dirty value flag[p626] must be set to true whenever the user interacts with the control in a way that changes the raw value[p608].

The cloning steps for textarea[p606] elements given *node*, *copy*, and *subtree* are to propagate the raw value[p608] and dirty value flag[p626] from *node* to *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 user validity to false, the dirty value flag back to false, and the raw value 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.

The **wrap** attribute is an enumerated attribute with the following keywords and states:

Keyword	State	Brief description
soft	**Soft**	Text is not to be wrapped when submitted (though can still be wrapped in the rendering).
hard	**Hard**	Text is to have newlines added by the user agent so that the text is wrapped when it is submitted.

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

If the element's wrap attribute is in the Hard state, the cols attribute must be specified.

For historical reasons, the element's value is normalized in three different ways for three different purposes. The raw value is the value as it was originally set. It is not normalized. The API value is the value used in the value IDL attribute, textLength IDL attribute, and by the maxlength and minlength content attributes. It is normalized so that line breaks use U+000A LINE FEED (LF) characters. Finally, there is the value, as used in form submission and other processing models in this specification. It is normalized as for the API value, and in addition, if necessary given the element's wrap attribute, additional line breaks are inserted to wrap the text at the given width.

The algorithm for obtaining the element's API value is to return the element's raw value, with newlines normalized.

The element's value is defined to be the element's API value with the textarea wrapping transformation applied. The **textarea wrapping transformation** is the following algorithm, as applied to a string:

1. If the element's wrap attribute is in the Hard state, insert U+000A LINE FEED (LF) characters into the string using an implementation-defined algorithm so that each line has no more than character width characters. For the purposes of this requirement, lines are delimited by the start of the string, the end of the string, and U+000A LINE FEED (LF) characters.

The **maxlength** attribute is a form control maxlength attribute.

If the textarea element has a maximum allowed value length, 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 less than or equal to the element's maximum allowed value length.

The **minlength** attribute is a form control minlength attribute.

The **required** attribute is a boolean attribute[p79]. When specified, the user will be required to enter a value before submitting the form.

Constraint validation: If the element has its required[p610] attribute specified, and the element is mutable[p626], and the element's value[p626] is the empty string, then the element is suffering from being missing[p651].

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[p610] attribute should not be used as an alternative to a label[p536]. For a longer hint or other advisory text, the title[p164] attribute is more appropriate.

Note

These mechanisms are very similar but subtly different: the hint given by the control's label[p536] is shown at all times; the short hint given in the placeholder[p610] attribute is shown before the user enters a value; and the hint in the title[p164] attribute is shown when the user requests further help.

User agents should present this hint to the user when the element's value[p626] is the empty string and the control is not focused[p875] (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[p875], then the user agent should nonetheless show the hint for the control if it was focused as a result of the autofocus[p887] attribute, since in that case the user will not have had an opportunity to examine the control before focusing it.

The name[p628] attribute represents the element's name. The dirname[p629] attribute controls how the element's directionality[p168] is submitted. The disabled[p630] attribute is used to make the control non-interactive and to prevent its value from being submitted. The form[p627] attribute is used to explicitly associate the textarea[p606] element with its form owner[p626]. The autocomplete[p633] attribute controls how the user agent provides autofill behavior.

For web developers (non-normative)

***textarea*.type**[p610]

Returns the string "textarea".

***textarea*.value**[p610]

Returns the current value of the element.

Can be set, to change the value.

The **type** IDL attribute must return the value "textarea".

The **defaultValue** attribute's getter must return the element's child text content.

The defaultValue[p610] 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[p626]. On setting, it must perform the following steps:

1. Let *oldAPIValue* be this element's API value[p626].
2. Set this element's raw value[p608] to the new value.
3. Set this element's dirty value flag[p626] to true.
4. If the new API value[p626] is different from *oldAPIValue*, then move the text entry cursor position[p647] to the end of the text control, unselecting any selected text and resetting the selection direction[p648] to "none".

The **textLength** IDL attribute must return the length of the element's API value[p626].

The willValidate[p654], validity[p654], and validationMessage[p656] IDL attributes, and the checkValidity()[p656], reportValidity()[p656], and setCustomValidity()[p654] methods, are part of the constraint validation API[p653]. The labels[p538] IDL attribute provides a list of the element's label[p536]s. The select()[p648], selectionStart[p648], selectionEnd[p649], selectionDirection[p649], setRangeText()[p650], and setSelectionRange()[p649] methods and IDL attributes expose the element's text selection. The disabled[p607], form[p628], and name[p607] IDL attributes are part of the element's forms API.

Example

Here is an example of a textarea[p606] being used for unrestricted free-form text input in a form:

```
<p>If you have any comments, please let us know: <textarea cols=80 name=comments></textarea></p>
```

To specify a maximum length for the comments, one can use the maxlength[p609] attribute:

```
<p>If you have any short comments, please let us know: <textarea cols=80 name=comments
maxlength=200></textarea></p>
```

To give a default value, text can be included inside the element:

```
<p>If you have any comments, please let us know: <textarea cols=80 name=comments>You
rock!</textarea></p>
```

You can also give a minimum length. Here, a letter needs to be filled out by the user; a template (which is shorter than the minimum length) is provided, but is insufficient to submit the form:

```
<textarea required minlength="500">Dear Madam Speaker,

Regarding your letter dated ...

...

Yours Sincerely,

...</textarea>
```

A placeholder can be given as well, to suggest the basic form to the user, without providing an explicit template:

```
<textarea placeholder="Dear Francine,

They closed the parks this week, so we won't be able to
meet your there. Should we just have dinner?

Love,
Daddy"></textarea>
```

To have the browser submit the directionality[p168] of the element along with the value, the dirname[p629] attribute can be specified:

```
<p>If you have any comments, please let us know (you may use either English or Hebrew for your
comments):
<textarea cols=80 name=comments dirname=comments.dir></textarea></p>
```

4.10.12 The output element §[p61] 1

Categories[p154]:

Flow content[p157].
Phrasing content[p157].
Listed[p532], labelable[p532], resettable[p532], and autocapitalize-and-autocorrect inheriting[p532] form-associated element[p532].
Palpable content[p158].

Contexts in which this element can be used[p154]:

Where phrasing content[p157] is expected.

Content model[p154]:

Phrasing content[p157].

Tag omission in text/html[p154]**:**

Neither tag is omissible.

Content attributes[p154]**:**

Global attributes[p162]
`for`[p612] — Specifies controls from which the output was calculated
`form`[p627] — Associates the element with a `form`[p532] element
`name`[p628] — Name of the element to use in the `form.elements`[p534] API.

Accessibility considerations[p154]**:**

For authors.
For implementers.

Sanitization[p154]**:**

Uncategorized[p1249].

DOM interface[p154]**:**

IDL

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

  [SameObject, PutForwards=value, Reflect="for"] readonly attribute DOMTokenList htmlFor;
  readonly attribute HTMLFormElement? form;
  [CEReactions, Reflect] attribute DOMString name;

  readonly attribute DOMString type;
  [CEReactions] attribute DOMString defaultValue;
  [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 NodeList labels;
};
```

The `output`[p611] element represents[p148] the result of a calculation performed by the application, or the result of a user action.

Note

This element can be contrasted with the `samp`[p299] 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`[p612] attribute, if specified, must contain a string consisting of an unordered set of unique space-separated tokens[p99], 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`[p627] attribute is used to explicitly associate the `output`[p611] element with its form owner[p626]. The `name`[p628] attribute represents the element's name. The `output`[p611] element is associated with a form so that it can be easily referenced[p148] 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[p612] is non-null, then return it.
2. Return this element's descendant text content.

The reset algorithm[p666] for output[p611] elements is to run these steps:

1. String replace all with this element's default value[p612] within this element.
2. Set this element's default value override[p612] to null.

For web developers (non-normative)

***output*.value[p613] [= *value*]**

Returns the element's current value.

Can be set, to change the value.

***output*.defaultValue[p613] [= *value*]**

Returns the element's current default value.

Can be set, to change the default value.

***output*.type[p613]**

Returns the string "output".

The **value** getter steps are to return this's descendant text content.

The value[p613] setter steps are:

1. Set this's default value override[p612] to its default value[p612].
2. String replace all with the given value within this.

The **defaultValue** getter steps are to return the result of running this's default value[p612].

The defaultValue[p613] setter steps are:

1. If this's default value override[p612] is null, then string replace all with the given value within this and return.
2. Set this's default value override[p612] to the given value.

The **type** getter steps are to return "output".

The willValidate[p654], validity[p654], and validationMessage[p656] IDL attributes, and the checkValidity()[p656], reportValidity()[p656], and setCustomValidity()[p654] methods, are part of the constraint validation API[p653]. The labels[p538] IDL attribute provides a list of the element's label[p536]s. The form[p628] and name[p612] IDL attributes are part of the element's forms API.

Example

A simple calculator could use output[p611] for its display of calculated results:

```
<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[p611] element is used to report the results of a calculation performed by a remote server, as they come in:

```
<output id="result"></output>
<script>
 var primeSource = new WebSocket('ws://primes.example.net/');
 primeSource.onmessage = function (event) {
   document.getElementById('result').value = event.data;
 }
</script>
```

4.10.13 The progress element §[p614]

Categories[p154]:

Flow content[p157].
Phrasing content[p157].
Labelable element[p532].
Palpable content[p158].

Contexts in which this element can be used[p154]:

Where phrasing content[p157] is expected.

Content model[p154]:

Phrasing content[p157], but there must be no progress[p614] element descendants.

Tag omission in text/html[p154]:

Neither tag is omissible.

Content attributes[p154]:

Global attributes[p162]
value[p614] — Current value of the element
max[p614] — Upper bound of range

Accessibility considerations[p154]:

For authors.
For implementers.

Sanitization[p154]:

Uncategorized[p1249].

DOM interface[p154]:

IDL

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

  [CEReactions, ReflectSetter] attribute double value;
  [CEReactions, ReflectPositive, ReflectDefault=1.0] attribute double max;
  readonly attribute double position;
  readonly attribute NodeList labels;
};
```

The progress[p614] element represents[p148] 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[p614] attribute with the current progress (either a number from 0.0 to 1.0, or, if the max[p614] attribute is specified, a number from 0 to the value of the max[p614] attribute). To make an indeterminate progress bar, remove the value[p614] 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`[p614] and `max`[p614] attributes, when present, must have values that are valid floating-point numbers[p81]. The `value`[p614] attribute, if present, must have a value greater than or equal to zero, and less than or equal to the value of the `max`[p614] attribute, if present, or 1.0, otherwise. The `max`[p614] attribute, if present, must have a value greater than zero.

Note

The `progress`[p614] element is the wrong element to use for something that is just a gauge, as opposed to task progress. For instance, indicating disk space usage using `progress`[p614] would be inappropriate. Instead, the `meter`[p616] element is available for such use cases.

User agent requirements: If the `value`[p614] 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`[p614] attribute, the user agent must parse the `max`[p614] attribute's value according to the rules for parsing floating-point number values[p82]. 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`[p614] 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[p615] of the progress bar is 1.0.

If the progress bar is a determinate progress bar, user agents must parse the `value`[p614] attribute's value according to the rules for parsing floating-point number values[p82]. If this does not result in an error and the parsed value is greater than zero, then the **value** of the progress bar is that parsed value. Otherwise, if parsing the `value`[p614] attribute's value resulted in an error or a number less than or equal to zero, then the value[p615] of the progress bar is zero.

If the progress bar is a determinate progress bar, then the **current value** is the maximum value[p615], if value[p615] is greater than the maximum value[p615], and value[p615] otherwise.

UA requirements for showing the progress bar: When representing a `progress`[p614] 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[p615] relative to the maximum value[p615].

For web developers (non-normative)

***progress*.position**[p615]

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[p615] by the maximum value[p615].

The **value** getter steps are to return 0 if this is an indeterminate progress bar; otherwise this's current value[p615].

Note

Setting the value[p615] 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 labels[p538] IDL attribute provides a list of the element's label[p536]s.

4.10.14 The meter element §[p616]

Categories[p154]:

Flow content[p157].
Phrasing content[p157].
Labelable element[p532].
Palpable content[p158].

Contexts in which this element can be used[p154]:

Where phrasing content[p157] is expected.

Content model[p154]:

Phrasing content[p157], but there must be no meter[p616] element descendants.

Tag omission in text/html[p154]:

Neither tag is omissible.

Content attributes[p154]:

Global attributes[p162]
value[p617] — Current value of the element
min[p617] — Lower bound of range
max[p617] — Upper bound of range
low[p617] — High limit of low range
high[p617] — Low limit of high range
optimum[p617] — Optimum value in gauge

Accessibility considerations[p154]:

For authors.
For implementers.

Sanitization[p154]:

Uncategorized[p1249].

DOM interface[p154]:

IDL

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

  [CEReactions, ReflectSetter] attribute double value;
  [CEReactions, ReflectSetter] attribute double min;
  [CEReactions, ReflectSetter] attribute double max;
  [CEReactions, ReflectSetter] attribute double low;
  [CEReactions, ReflectSetter] attribute double high;
  [CEReactions, ReflectSetter] attribute double optimum;
  readonly attribute NodeList labels;
};
```

The meter[p616] element represents[p148] 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[p616] element should not be used to indicate progress (as in a progress bar). For that role, HTML provides a separate progress[p614] element.

Note

The meter[p616] 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[p617] attribute must be specified. The value[p617], min[p617], low[p617], high[p617], max[p617], and optimum[p617] attributes, when present, must have values that are valid floating-point numbers[p81].

In addition, the attributes' values are further constrained:

Let *value* be the value[p617] attribute's number.

If the min[p617] attribute is specified, then let *minimum* be that attribute's value; otherwise, let it be 0.

If the max[p617] attribute is specified, then let *maximum* be that attribute's value; otherwise, let it be 1.0.

The following inequalities must hold, as applicable:

- *minimum* ≤ *value* ≤ *maximum*
- *minimum* ≤ low[p617] ≤ *maximum* (if low[p617] is specified)
- *minimum* ≤ high[p617] ≤ *maximum* (if high[p617] is specified)
- *minimum* ≤ optimum[p617] ≤ *maximum* (if optimum[p617] is specified)
- low[p617] ≤ high[p617] (if both low[p617] and high[p617] are specified)

Note

If no minimum or maximum is specified, then the range is assumed to be 0..1, and the value thus has to be within that range.

Authors are encouraged to include a textual representation of the gauge's state in the element's contents, for users of user agents that do not support the meter[p616] element.

When used with microdata[p826], the meter[p616] element's value[p617] attribute provides the element's machine-readable value.

Example

The following examples show three gauges that would all be three-quarters full:

```
Storage space usage: <meter value=6 max=8>6 blocks used (out of 8 total)</meter>
```

```
Voter turnout: <meter value=0.75><img alt="75%" src="graph75.png"></meter>
```

```
Tickets sold: <meter min="0" max="100" value="75"></meter>
```

The following example is incorrect use of the element, because it doesn't give a range (and since the default maximum is 1, both of the gauges would end up looking maxed out):

```
<p>The grapefruit pie had a radius of <meter value=12>12cm</meter>
and a height of <meter value=2>2cm</meter>.</p> <!-- BAD! -->
```

Instead, one would either not include the meter element, or use the meter element with a defined range to give the dimensions in context compared to other pies:

```
<p>The grapefruit pie had a radius of 12cm and a height of
2cm.</p>
```

```
<dl>
 <dt>Radius: <dd> <meter min=0 max=20 value=12>12cm</meter>
 <dt>Height: <dd> <meter min=0 max=10 value=2>2cm</meter>
</dl>
```

There is no explicit way to specify units in the meter[p616] element, but the units may be specified in the title[p164] attribute in free-form text.

Example

The example above could be extended to mention the units:

```
<dl>
 <dt>Radius: <dd> <meter min=0 max=20 value=12 title="centimeters">12cm</meter>
 <dt>Height: <dd> <meter min=0 max=10 value=2 title="centimeters">2cm</meter>
</dl>
```

User agent requirements: User agents must parse the min[p617], max[p617], value[p617], low[p617], high[p617], and optimum[p617] attributes using the rules for parsing floating-point number values[p82].

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[p617] 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[p617] 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[p617] 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[p617] 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[p617] 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[p617] 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.

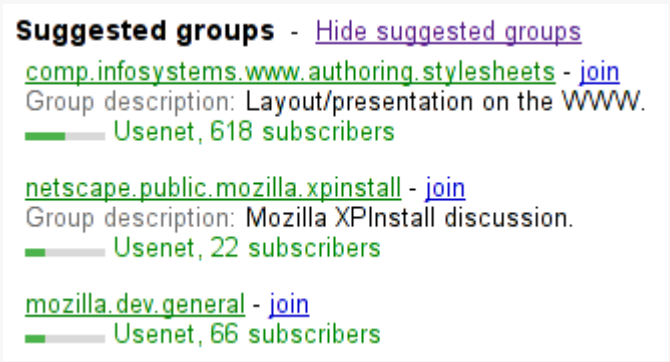
UA requirements for showing the gauge: When representing a meter[p616] element to the user, the UA should indicate the relative position of the actual value to the minimum and maximum values, and the relationship between the actual value and the three regions of the gauge.

Example

The following markup:

```
<h3>Suggested groups</h3>
<menu>
 <li><a href="?cmd=hsg" onclick="hideSuggestedGroups()">Hide suggested groups</a></li>
</menu>
<ul>
 <li>
  <p><a href="/group/comp.infosystems.www.authoring.stylesheets/
view">comp.infosystems.www.authoring.stylesheets</a> -
     <a href="/group/comp.infosystems.www.authoring.stylesheets/subscribe">join</a></p>
  <p>Group description: <strong>Layout/presentation on the WWW.</strong></p>
  <p><meter value="0.5">Moderate activity,</meter> Usenet, 618 subscribers</p>
 </li>
 <li>
  <p><a href="/group/netscape.public.mozilla.xpinstall/view">netscape.public.mozilla.xpinstall</a>
-
     <a href="/group/netscape.public.mozilla.xpinstall/subscribe">join</a></p>
  <p>Group description: <strong>Mozilla XPInstall discussion.</strong></p>
  <p><meter value="0.25">Low activity,</meter> Usenet, 22 subscribers</p>
 </li>
 <li>
  <p><a href="/group/mozilla.dev.general/view">mozilla.dev.general</a> -
     <a href="/group/mozilla.dev.general/subscribe">join</a></p>
  <p><meter value="0.25">Low activity,</meter> Usenet, 66 subscribers</p>
 </li>
</ul>
```

Might be rendered as follows:



User agents may combine the value of the `title`[p164] attribute and the other attributes to provide context-sensitive help or inline text detailing the actual values.

Example

For example, the following snippet:

```
<meter min=0 max=60 value=23.2 title=seconds></meter>
```

...might cause the user agent to display a gauge with a tooltip saying "Value: 23.2 out of 60." on one line and "seconds" on a second line.

The **`value`** getter steps are to return this's actual value[p618].

The **`min`** getter steps are to return this's minimum value[p618].

The **`max`** getter steps are to return this's maximum value[p618].

The **`low`** getter steps are to return this's low boundary[p618].

The **`high`** getter steps are to return this's high boundary[p618].

The **`optimum`** getter steps are to return this's optimum value[p619].

The `labels`[p538] IDL attribute provides a list of the element's `label`[p536]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>170 261 928 bytes used
out of 233 257 824 bytes available</meter></p>
```

4.10.15 The `fieldset` element §[p62]

Categories[p154]:

Flow content[p157].
Listed[p532] and autocapitalize-and-autocorrect inheriting[p532] form-associated element[p532].
Palpable content[p158].

Contexts in which this element can be used[p154]:

Where flow content[p157] is expected.

Content model[p154]**:**

Optionally, a legend[p623] element, followed by flow content[p157].

Tag omission in text/html[p154]**:**

Neither tag is omissible.

Content attributes[p154]**:**

Global attributes[p162]
disabled[p621] — Whether the descendant form controls, except any inside legend[p623], are disabled
form[p627] — Associates the element with a form[p532] element
name[p628] — Name of the element to use in the form.elements[p534] API.

Accessibility considerations[p154]**:**

For authors.
For implementers.

Sanitization[p154]**:**

Uncategorized[p1249].

DOM interface[p154]**:**

IDL

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

  [CEReactions, Reflect] attribute boolean disabled;
  readonly attribute HTMLFormElement? form;
  [CEReactions, Reflect] attribute DOMString name;

  readonly attribute DOMString type;

  [SameObject] readonly attribute HTMLCollection elements;

  readonly attribute boolean willValidate;
  [SameObject] readonly attribute ValidityState validity;
  readonly attribute DOMString validationMessage;
  boolean checkValidity();
  boolean reportValidity();
  undefined setCustomValidity(DOMString error);
};
```

The fieldset[p620] element represents[p148] a set of form controls (or other content) grouped together, optionally with a caption. The caption is given by the first legend[p623] element that is a child of the fieldset[p620] 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[p620] element, excluding those that are descendants of the fieldset[p620] element's first legend[p623] element child, if any, to be disabled[p630].

A fieldset[p620] element is a **disabled fieldset** if it matches any of the following conditions:

- Its disabled[p621] attribute is specified.
- It is a descendant of another fieldset[p620] element whose disabled[p621] attribute is specified, and it is *not* a descendant of that fieldset[p620] element's first legend[p623] element child, if any.

The form[p627] attribute is used to explicitly associate the fieldset[p620] element with its form owner[p626]. The name[p628] attribute represents the element's name.

For web developers (non-normative)

***fieldset*.type**[p622]

Returns the string "fieldset".

***fieldset*.elements**[p622]

Returns an HTMLCollection of the form controls in the element.

The **type** IDL attribute must return the string "fieldset".

The **elements** IDL attribute must return an HTMLCollection rooted at the fieldset[p620] element, whose filter matches listed elements[p532].

The willValidate[p654], validity[p654], and validationMessage[p656] attributes, and the checkValidity()[p656], reportValidity()[p656], and setCustomValidity()[p654] methods, are part of the constraint validation API[p653]. The form[p628] and name[p621] IDL attributes are part of the element's forms API.

Example

This example shows a fieldset[p620] element being used to group a set of related controls:

```
<fieldset>
 <legend>Display</legend>
 <p><label><input type=radio name=c value=0 checked> Black on White</label>
 <p><label><input type=radio name=c value=1> White on Black</label>
 <p><label><input type=checkbox name=g> Use grayscale</label>
 <p><label>Enhance contrast <input type=range name=e list=contrast min=0 max=100 value=0
step=1></label>
 <datalist id=contrast>
  <option label=Normal value=0>
  <option label=Maximum value=100>
 </datalist>
</fieldset>
```

Example

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> <label>
  <input type=checkbox name=club onchange="form.clubfields.disabled = !checked">
  Use Club Card
 </label> </legend>
 <p><label>Name on card: <input name=clubname required></label></p>
 <p><label>Card number: <input name=clubnum required pattern="[-0-9]+"></label></p>
 <p><label>Expiry date: <input name=clubexp type=month></label></p>
</fieldset>
```

Example

You can also nest fieldset[p620] elements. Here is an example expanding on the previous one that does so:

```
<fieldset name="clubfields" disabled>
 <legend> <label>
  <input type=checkbox name=club onchange="form.clubfields.disabled = !checked">
  Use Club Card
 </label> </legend>
 <p><label>Name on card: <input name=clubname required></label></p>
 <fieldset name="numfields">
  <legend> <label>
   <input type=radio checked name=clubtype onchange="form.numfields.disabled = !checked">
   My card has numbers on it
  </label> </legend>
  <p><label>Card number: <input name=clubnum required pattern="[-0-9]+"></label></p>
```

```
 </fieldset>
 <fieldset name="letfields" disabled>
  <legend> <label>
   <input type=radio name=clubtype onchange="form.letfields.disabled = !checked">
   My card has letters on it
  </label> </legend>
  <p><label>Card code: <input name=clublet required pattern="[A-Za-z]+"></label></p>
 </fieldset>
</fieldset>
```

In this example, if the outer "Use Club Card" checkbox is not checked, everything inside the outer `fieldset`[p620], including the two radio buttons in the legends of the two nested `fieldset`[p620]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`[p620]s is to be enabled.

Example

This example shows a grouping of controls where the `legend`[p623] element both labels the grouping, and the nested heading element surfaces the grouping in the document outline:

```
<fieldset>
 <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>
</fieldset>
```

4.10.16 The `legend` element §[p623]

Categories[p154]**:**

None.

Contexts in which this element can be used[p154]**:**

As the first child of a `fieldset`[p620] element.
As the first child of an `optgroup`[p600] element.

Content model[p154]**:**

If the element is a child of an `optgroup`[p600] element: Phrasing content[p157], but there must be no interactive content[p158] and no descendant with the `tabindex`[p878] attribute.
Otherwise: Phrasing content[p157], optionally intermixed with heading content[p157].

Tag omission in text/html[p154]**:**

Neither tag is omissible.

Content attributes[p154]**:**

Global attributes[p162]

Accessibility considerations[p154]**:**

For authors.

For implementers.

Sanitization[p154]:

Uncategorized[p1249].

DOM interface[p154]:

IDL

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

  readonly attribute HTMLFormElement? form;

  // also has obsolete members
};
```

The legend[p623] element represents[p148] a caption for the rest of the contents of the legend[p623] element's parent fieldset[p620] element, if any. Otherwise, if the legend[p623] has a parent optgroup[p600] element, then the legend[p623] represents the optgroup[p600]'s label.

For web developers (non-normative)

***legend*.form[p624]**

Returns the element's form[p532] element, if any, or null otherwise.

The **form** IDL attribute's behavior depends on whether the legend[p623] element is in a fieldset[p620] element or not. If the legend[p623] has a fieldset[p620] element as its parent, then the form[p624] IDL attribute must return the same value as the form[p628] IDL attribute on that fieldset[p620] element. Otherwise, it must return null.

4.10.17 The selectedcontent element §[p624]

Categories[p154]:

Phrasing content[p157].

Contexts in which this element can be used[p154]:

As a descendant of a button[p585] element which is a child of a select[p590] element.

Content model[p154]:

Nothing[p155].

Tag omission in text/html[p154]:

Neither tag is omissible.

Content attributes[p154]:

Global attributes[p162]

Accessibility considerations[p154]:

For authors.
For implementers.

Sanitization[p154]:

Uncategorized[p1249].

DOM interface[p154]:

IDL

```
[Exposed=Window]
interface HTMLSelectedContentElement : HTMLElement {
  [HTMLConstructor] constructor();
};
```

The selectedcontent[p624] element reflects the contents of a select[p590] element's currently selected option[p601] element. selectedcontent[p624] elements can be used to declaratively show the selected option[p601] element's contents within the select[p590] element's first child button[p585] element.

Note

The option element's label attribute can be used to render a visible label for the option, but the selectedcontent element will not reflect the content of the label attribute.

Every selectedcontent element has a boolean **disabled**, which is initially false.

To **update a select's descendant selectedcontent elements** given a select element *select*:

1. If *select* has the multiple attribute, then return.
2. Let *descendantSelectedcontents* be a list of *select*'s descendants, in tree order, that are selectedcontent elements whose disabled is false.
3. For each *selectedcontent* of *descendantSelectedcontents*: update a selectedcontent given *select* and *selectedcontent*.

To **update a selectedcontent** given a select element *select* and a selectedcontent element *selectedcontent*:

1. If *selectedcontent*'s disabled is true, then return.
2. Let *option* be the first option in *select*'s list of options whose selectedness is true, if any such option exists; otherwise null.
3. If *option* is null, then clear a selectedcontent given *selectedcontent*.
4. Otherwise, clone an option into a selectedcontent given *option* and *selectedcontent*.

To **clone an option into a selectedcontent**, given an option element *option* and a selectedcontent element *selectedcontent*:

1. Let *documentFragment* be the result of creating a document fragment given *option*'s node document.
2. For each *child* of *option*'s children:
    1. Let *childClone* be the result of running clone given *child* with subtree set to true.
    2. Append *childClone* to *documentFragment*.
3. Replace all with *documentFragment* within *selectedcontent*.

To **clear a selectedcontent** given a selectedcontent element *selectedcontent*:

1. Replace all with null within *selectedcontent*.

The selectedcontent HTML element insertion steps, given *selectedcontent*, are to recalculate a selectedcontent element's disabledness given *selectedcontent*.

The selectedcontent HTML element post-connection steps, given *selectedcontent*, are:

1. Let *select* be the result of recalculate a selectedcontent element's disabledness given *selectedcontent*.
2. If *selectedcontent*'s disabled is true, *select* is null, or *select* has the multiple attribute, then return.
3. Update a selectedcontent given *select* and *selectedcontent*.

The selectedcontent HTML element moving steps, given *movedNode*, *isSubtreeRoot*, and *oldAncestor*, are:

1. Let *select* be the result of recalculate a selectedcontent element's disabledness given *movedNode*.
2. If *movedNode*'s disabled is true, *select* is null, or *select* has the multiple attribute, then return.
3. Queue a microtask to update a selectedcontent given *select* and *movedNode*.

To **recalculate a selectedcontent element's disabledness**, given a selectedcontent element *selectedcontent*, run these steps. They return a select element or null.

1. Let *nearestSelectAncestor* be null.
2. Set *selectedcontent*'s disabled to false.

3. For each *ancestor* of *selectedcontent*'s ancestors, in reverse tree order:

    1. If *ancestor* is a `select` element:

        1. If *nearestSelectAncestor* is null, then set *nearestSelectAncestor* to *ancestor* and continue.

        2. Set *selectedcontent*'s disabled to true and break.

    2. If *ancestor* is an `option` element or a `selectedcontent` element, then set *selectedcontent*'s disabled to true and break.

4. Return *nearestSelectAncestor*.

4.10.18 Form control infrastructure

4.10.18.1 A form control's value

Most form controls have a **value** and a **checkedness**. (The latter is only used by `input` elements.) These are used to describe how the user interacts with the control.

A control's value is its internal state. As such, it might not match the user's current input.

Example

For instance, if a user enters the word "`three`" into a numeric field that expects digits, the user's input would be the string "three" but the control's value would remain unchanged. Or, if a user enters the email address " `awesome@example.com`" (with leading whitespace) into an email field, the user's input would be the string " awesome@example.com" but the browser's UI for email fields might translate that into a value of "`awesome@example.com`" (without the leading whitespace).

`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.

Some form controls also have an **optional value**. This largely mirrors the value but doesn't normalize to an empty string.

Note *This ought to be used sparingly, you generally want value.*

`input`, `textarea`, and `select` elements have a **user validity** boolean. It is initially set to false.

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.

4.10.18.2 Mutability

A form control can be designated as **mutable**.

Note

This determines (by means of definitions and requirements in this specification that rely on whether an element is so designated) whether or not the user can modify the value or checkedness of a form control, or whether or not a control can be automatically prefilled.

4.10.18.3 Association of controls and forms

A form-associated element can have a relationship with a `form` element, which is called the element's **form owner**. If a form-associated element is not associated with a `form` element, its form owner is said to be null.

A form-associated element[p532] has an associated **parser inserted flag**.

A form-associated element[p532] is, by default, associated with its nearest ancestor `form`[p532] element (as described below), but, if it is listed[p532], may have a **`form`** attribute specified to override this.

Note

This feature allows authors to work around the lack of support for nested `form`[p532] elements.

If a listed[p532] form-associated element[p532] has a `form`[p627] attribute specified, then that attribute's value must be the ID of a `form`[p532] 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`[p532] 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[p532] being associated with a `form`[p532] element that is not its ancestor.

When a form-associated element[p532] is created, its form owner[p626] must be initialized to null (no owner).

When a form-associated element[p532] is to be **associated** with a form, its form owner[p626] must be set to that form.

When a listed[p532] form-associated element[p532]'s `form`[p627] attribute is set, changed, or removed, then the user agent must reset the form owner[p627] of that element.

When a listed[p532] form-associated element[p532] has a `form`[p627] attribute and the ID of any of the elements in the tree changes, then the user agent must reset the form owner[p627] of that form-associated element[p532].

When a listed[p532] form-associated element[p532] has a `form`[p627] attribute and an element with an ID is inserted into[p47] or removed from[p47] the `Document`[p135], or its HTML element moving steps[p46] are run, then the user agent must reset the form owner[p627] of that form-associated element[p532].

Note

The form owner is also reset by the HTML element insertion steps[p46], HTML element removing steps[p46], and HTML element moving steps[p46].

To **reset the form owner** of a form-associated element[p532] *element*:

1. Unset *element*'s parser inserted flag[p627].
2. If all of the following are true:
   - *element*'s form owner[p626] is not null;
   - *element* is not listed[p532] or its `form`[p627] content attribute is not present; and
   - *element*'s form owner[p626] is its nearest `form`[p532] element ancestor after the change to the ancestor chain,

   then return.
3. Set *element*'s form owner[p626] to null.
4. If *element* is listed[p532], has a `form`[p627] content attribute, and is connected:
   1. If the first element in *element*'s tree, in tree order, to have an ID that is identical to *element*'s `form`[p627] content attribute's value, is a `form`[p532] element, then associate[p627] the *element* with that `form`[p532] element.
5. Otherwise, if *element* has an ancestor `form`[p532] element, then associate[p627] *element* with the nearest such ancestor `form`[p532] 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[p626] of "d" would be the inner nested form "c", while the form owner[p626] of "e" would be the outer form "a".

This happens as follows: First, the "e" node gets associated with "c" in the HTML parser[p1368]. Then, the `innerHTML`[p1229] 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`[p532] elements, and there is a parse error[p1370] for the `</form>` tag.

For web developers (non-normative)

***element*.form**[p628]

Returns the element's form owner[p626].

Returns null if there isn't one.

Listed[p532] form-associated elements[p532] except for form-associated custom elements[p797] have a **form** IDL attribute, which, on getting, must return the element's form owner[p626], or null if there isn't one.

Form-associated custom elements[p797] don't have a `form`[p628] IDL attribute. Instead, their `ElementInternals`[p809] object has a **form** IDL attribute. On getting, it must throw a `"NotSupportedError"` `DOMException` if the target element[p810] is not a form-associated custom element[p797]. Otherwise, it must return the element's form owner[p626], or null if there isn't one.

4.10.19 Attributes common to form controls §[p628]

4.10.19.1 Naming form controls: the `name`[p628] attribute §[p628]

The **name** content attribute gives the name of the form control, as used in form submission[p657] and in the `form`[p532] element's `elements`[p534] 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`[p533] is allowed. An ASCII case-insensitive match for the name **_charset_** is special: if used as the name of a Hidden[p546] control with no `value`[p543] attribute, then during submission the `value`[p543] attribute is automatically given a value consisting of the submission character encoding.

Note

DOM clobbering is a common cause of security issues. Avoid using the names of built-in form properties with the `name`[p628] content attribute.

In this example, the `input`[p539] element overrides the built-in `method`[p631] 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.*

4.10.19.2 Submitting element directionality: the dirname attribute

The **dirname** attribute on a form control element enables the submission of the directionality of the element, and gives the name of the 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:

```
<form action="addcomment.cgi" method=post>
 <p><label>Comment: <input type=text name="comment" dirname="comment.dir" required></label></p>
 <p><button name="mode" type=submit value="add">Post Comment</button></p>
</form>
```

When the user submits the form, the user agent includes three fields, one called "comment", one called "comment.dir", and one called "mode"; so if the user types "Hello", the submission body might be something like:

```
comment=Hello&comment.dir=ltr&mode=add
```

If the user manually switches to a right-to-left writing direction and enters "مرحبا", the submission body might be something like:

```
comment=%D9%85%D8%B1%D8%AD%D8%A8%D8%A7&comment.dir=rtl&mode=add
```

4.10.19.3 Limiting user input length: the maxlength attribute

A **form control maxlength attribute**, controlled by 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).*

4.10.19.4 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).

Note

The `minlength` attribute does not imply the `required` attribute. If the form control has no `required` attribute, then the value can still be omitted; the `minlength` attribute only kicks in once the user has entered a value at all. If the empty string is not allowed, then the `required` attribute also needs to be set.

If an element has its form control `minlength` attribute specified, the attribute's value must be a valid non-negative integer. If the attribute is specified and applying the rules for parsing non-negative integers to its value results in a number, then that number is the element's **minimum allowed value length**. If the attribute is omitted or parsing its value results in an error, then there is no minimum allowed value length.

If an element has both a maximum allowed value length and a minimum allowed value length, the minimum allowed value length must be smaller than or equal to the maximum allowed value length.

Constraint validation: If an element has a minimum allowed value length, its dirty value flag is true, its value was last changed by a user edit (as opposed to a change made by a script), its value is not the empty string, and the 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.

```
<form action="/events/menu.cgi" method="post">
 <p><label>Name of Event: <input required minlength=5 maxlength=50 name=event></label></p>
 <p><label>Describe what you would like for breakfast, if anything:
    <textarea name="breakfast" minlength="10"></textarea></label></p>
 <p><label>Describe what you would like for lunch, if anything:
    <textarea name="lunch" minlength="10"></textarea></label></p>
 <p><label>Describe what you would like for dinner, if anything:
    <textarea name="dinner" minlength="10"></textarea></label></p>
 <p><input type=submit value="Submit Request"></p>
</form>
```

4.10.19.5 Enabling and disabling form controls: the `disabled` attribute

The **`disabled`** content attribute is a boolean attribute.

✔ MDN

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 are true:

- 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); or
- the element is a descendant of a `fieldset` element whose `disabled` attribute is specified, and the element 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.

Note

Being disabled[p630] does not prevent all modifications to the form control. For example, the control's value[p626] or checkedness[p626] could be modified programmatically from JavaScript. Or, they could be indirectly modified by user action, e.g., if other non-disabled elements in the control's radio button group[p562] were modified.

Constraint validation: If an element is disabled[p630], it is barred from constraint validation[p651].

4.10.19.6 Form submission attributes §[p63 1]

Attributes for form submission can be specified both on form[p532] elements and on submit buttons[p532] (elements that represent buttons that submit forms, e.g. an input[p539] element whose type[p541] attribute is in the Submit Button[p565] state).

The attributes for form submission[p631] that may be specified on form[p532] elements are action[p631], enctype[p632], method[p631], novalidate[p632], and target[p632].

The corresponding attributes for form submission[p631] that may be specified on submit buttons[p532] are formaction[p631], formenctype[p632], formmethod[p631], formnovalidate[p632], and formtarget[p632]. When omitted, they default to the values given on the corresponding attributes on the form[p532] element.

The **action** and **formaction** content attributes, if specified, must have a value that is a valid non-empty URL potentially surrounded by spaces[p100].

The **action** of an element is the value of the element's formaction[p631] attribute, if the element is a submit button[p532] and has such an attribute, or the value of its form owner[p626]'s action[p631] attribute, if *it* has one, or else the empty string.

The **method** and **formmethod** content attributes are enumerated attributes[p80] with the following keywords and states:

Keyword	State	Brief description
get	**GET**	Indicates the form[p532] will use the HTTP GET method.
post	**POST**	Indicates the form[p532] will use the HTTP POST method.
dialog	**Dialog**	Indicates the form[p532] is intended to close the dialog[p675] box in which the form finds itself, if any, and otherwise not submit.

The method[p631] attribute's *missing value default*[p80] and *invalid value default*[p80] are both the GET[p631] state.

The formmethod[p631] attribute has no *missing value default*[p80], and its *invalid value default*[p80] is the GET[p631] state.

The **method** of an element is one of those states. If the element is a submit button[p532] and has a formmethod[p631] attribute, then the element's method[p631] is that attribute's state; otherwise, it is the form owner[p626]'s method[p631] attribute's state.

Example

Here the method[p631] attribute is used to explicitly specify the default value, "get[p631]", 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[p631] attribute is used to specify the value "post[p631]", 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[p532] is used with a dialog[p675]. The method[p631] attribute's "dialog[p631]" 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') {
     // ...
   } else {
     // ...
   }
 };
</script>
```

The **enctype** and **formenctype** content attributes are enumerated attributes[p80] 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 attribute's *missing value default*[p80] and *invalid value default*[p80] are both the application/x-www-form-urlencoded[p632] state.

The formenctype[p632] attribute has no *missing value default*[p80], and its *invalid value default*[p80] is the application/x-www-form-urlencoded[p632] state.

The **enctype** of an element is one of those three states. If the element is a submit button[p532] and has a formenctype[p632] attribute, then the element's enctype[p632] is that attribute's state; otherwise, it is the form owner[p626]'s enctype[p632] attribute's state.

The **target** and **formtarget** content attributes, if specified, must have values that are valid navigable target names or keywords[p1043].

The **novalidate** and **formnovalidate** content attributes are boolean attributes[p79]. If present, they indicate that the form is not to be validated during submission.

The **no-validate state** of an element is true if the element is a submit button[p532] and the element's formnovalidate[p632] attribute is present, or if the element's form owner[p626]'s novalidate[p632] 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.

```
<form action="editor.cgi" method="post">
 <p><label>Name: <input required name=fn></label></p>
 <p><label>Essay: <textarea required name=essay></textarea></label></p>
 <p><input type=submit name=submit value="Submit essay"></p>
 <p><input type=submit formnovalidate name=save value="Save essay"></p>
```

```
 <p><input type=submit formnovalidate name=cancel value="Cancel"></p>
</form>
```

The **action** getter steps are:

1. Let *attribute* be this's action[p631] attribute.
2. If *attribute* is null or *attribute*'s value is the empty string, then return this's node document's URL.
3. Let *urlString* be the result of encoding-parsing-and-serializing a URL[p101] given *attribute*'s value, relative to this's node document.
4. If *urlString* is not failure, then return *urlString*.
5. Return *attribute*'s value, converted to a scalar value string.

The **formAction** getter steps are:

1. Let *attribute* be this's formaction[p631] attribute.
2. If *attribute* is null or *attribute*'s value is the empty string, then return this's node document's URL.
3. Let *urlString* be the result of encoding-parsing-and-serializing a URL[p101] given *attribute*'s value, relative to this's node document.
4. If *urlString* is not failure, then return *urlString*.
5. Return *attribute*'s value, converted to a scalar value string.

The **method** and **enctype** IDL attributes must reflect[p108] the respective content attributes of the same name, limited to only known values[p109]. The **encoding** IDL attribute must reflect[p108] the enctype[p632] content attribute, limited to only known values[p109]. The **formEnctype** IDL attribute must reflect[p108] the formenctype[p632] content attribute, limited to only known values[p109]. The **formMethod** IDL attribute must reflect[p108] the formmethod[p631] content attribute, limited to only known values[p109].

4.10.19.7 Autofill §[p633]

4.10.19.7.1 Autofilling form controls: the autocomplete[p633] attribute §[p633]

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[p633] attribute describes what input is expected from users. When wearing the **autofill anchor mantle**, the autocomplete[p633] attribute describes the meaning of the given value.

On an input[p539] element whose type[p541] attribute is in the Hidden[p546] state, the autocomplete[p633] attribute wears the autofill anchor mantle[p633]. In all other cases, it wears the autofill expectation mantle[p633].

When wearing the autofill expectation mantle[p633], the autocomplete[p633] attribute, if specified, must have a value that is an ordered set of space-separated tokens[p99] consisting of either a single token that is an ASCII case-insensitive match for the string "off[p635]", or a single token that is an ASCII case-insensitive match for the string "on[p635]", or autofill detail tokens[p633].

When wearing the autofill anchor mantle[p633], the autocomplete[p633] attribute, if specified, must have a value that is an ordered set of space-separated tokens[p99] consisting of just autofill detail tokens[p633] (i.e. the "on[p635]" and "off[p635]" 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:

```
<fieldset>
 <legend>Ship the blue gift to...</legend>
 <p> <label> Address:     <textarea name=ba autocomplete="section-blue shipping street-
address"></textarea> </label>
 <p> <label> City:        <input name=bc autocomplete="section-blue shipping address-
level2"> </label>
 <p> <label> Postal Code: <input name=bp autocomplete="section-blue shipping postal-code">
</label>
</fieldset>
<fieldset>
 <legend>Ship the red gift to...</legend>
 <p> <label> Address:     <textarea name=ra autocomplete="section-red shipping street-
address"></textarea> </label>
 <p> <label> City:        <input name=rc autocomplete="section-red shipping address-
level2"> </label>
 <p> <label> Postal Code: <input name=rp autocomplete="section-red shipping postal-code">
</label>
</fieldset>
```

2. Optionally, a token that is an ASCII case-insensitive match for one of the following strings:

   - "**shipping**", meaning the field is part of the shipping address or contact information
   - "**billing**", meaning the field is part of the billing address or contact information

3. Either of the following two options:

   - A token that is an ASCII case-insensitive match for one of the following autofill field[p636] names, excluding those that are inappropriate for the control[p636]:

     - "name[p636]"
     - "honorific-prefix[p636]"
     - "given-name[p636]"
     - "additional-name[p636]"
     - "family-name[p636]"
     - "honorific-suffix[p636]"
     - "nickname[p636]"
     - "username[p636]"
     - "new-password[p636]"
     - "current-password[p636]"
     - "one-time-code[p636]"
     - "organization-title[p636]"
     - "organization[p636]"
     - "street-address[p636]"
     - "address-line1[p636]"
     - "address-line2[p636]"
     - "address-line3[p636]"
     - "address-level4[p637]"
     - "address-level3[p637]"
     - "address-level2[p637]"
     - "address-level1[p637]"
     - "country[p637]"
     - "country-name[p637]"
     - "postal-code[p637]"
     - "cc-name[p637]"
     - "cc-given-name[p637]"
     - "cc-additional-name[p637]"
     - "cc-family-name[p637]"
     - "cc-number[p637]"
     - "cc-exp[p637]"
     - "cc-exp-month[p637]"
     - "cc-exp-year[p637]"
     - "cc-csc[p637]"
     - "cc-type[p637]"
     - "transaction-currency[p637]"
     - "transaction-amount[p637]"
     - "language[p637]"
     - "bday[p637]"
     - "bday-day[p637]"

- "bday-month[p637]"
- "bday-year[p637]"
- "sex[p637]"
- "url[p637]"
- "photo[p637]"

(See the table below for descriptions of these values.)

- The following, in the given order:

  1. Optionally, a token that is an ASCII case-insensitive match for one of the following strings:

     - "**home**", meaning the field is for contacting someone at their residence
     - "**work**", meaning the field is for contacting someone at their workplace
     - "**mobile**", meaning the field is for contacting someone regardless of location
     - "**fax**", meaning the field describes a fax machine's contact details
     - "**pager**", meaning the field describes a pager's or beeper's contact details

  2. A token that is an ASCII case-insensitive match for one of the following autofill field[p636] names, excluding those that are inappropriate for the control[p636]:

     - "tel[p637]"
     - "tel-country-code[p638]"
     - "tel-national[p638]"
     - "tel-area-code[p638]"
     - "tel-local[p638]"
     - "tel-local-prefix[p638]"
     - "tel-local-suffix[p638]"
     - "tel-extension[p638]"
     - "email[p638]"
     - "impp[p638]"

     (See the table below for descriptions of these values.)

4. Optionally, a token that is an ASCII case-insensitive match for the string "**webauthn**", meaning the user agent should show public key credentials available via `conditional` mediation when the user interacts with the form control. `webauthn`[p635] is only valid for `input`[p539] and `textarea`[p606] elements.

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[p633]...**

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[p636] listed above indicates 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`[p633] attribute is omitted, the default value corresponding to the state of the element's form owner[p626]'s `autocomplete`[p533] attribute is used instead (either "`on`[p635]" or "`off`[p635]"). If there is no form owner[p626], then the value "`on`[p635]" is used.

↪ **When wearing the autofill anchor mantle[p633]...**

The autofill field[p636] listed above indicates 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.

```
<form method=post action="step2.cgi">
 <input type=hidden autocomplete=transaction-currency value="CHF">
```

```
 <input type=hidden autocomplete=transaction-amount value="15.00">
 <p><label>Credit card number: <input type=text inputmode=numeric autocomplete=cc-
number></label>
 <p><label>Expiry Date: <input type=month autocomplete=cc-exp></label>
 <p><input type=submit value="Continue...">
</form>
```

The **autofill field** keywords relate to each other as described in the table below. Each field name listed on a row of this table corresponds to the meaning given in the cell for that row in the column labeled "Meaning". Some fields correspond to 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[p637]"), or as two fields, one giving the month ("cc-exp-month[p637]") and one the year ("cc-exp-year[p637]"). 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[p636] name is **inappropriate for a control** if the control does not belong to the group listed for that autofill field[p636] in the fifth column of the first row describing that autofill field[p636] in the table below. What controls fall into each group is described below the table.

Field name	Meaning	Canonical Format	Canonical Format Example	Control group
"name"	Full name	Free-form text, no newlines	Sir Timothy John Berners-Lee, OM, KBE, FRS, FREng, FRSA	Text[p638]
"honorific-prefix"	Prefix or title (e.g. "Mr.", "Ms.", "Dr.", "Mlle")	Free-form text, no newlines	Sir	Text[p638]
"given-name"	Given name (in some Western cultures, also known as the *first name*)	Free-form text, no newlines	Timothy	Text[p638]
"additional-name"	Additional names (in some Western cultures, also known as *middle names*, forenames other than the first name)	Free-form text, no newlines	John	Text[p638]
"family-name"	Family name (in some Western cultures, also known as the *last name* or *surname*)	Free-form text, no newlines	Berners-Lee	Text[p638]
"honorific-suffix"	Suffix (e.g. "Jr.", "B.Sc.", "MBASW", "II")	Free-form text, no newlines	OM, KBE, FRS, FREng, FRSA	Text[p638]
"nickname"	Nickname, screen name, handle: a typically short name used instead of the full name	Free-form text, no newlines	Tim	Text[p638]
"organization-title"	Job title (e.g. "Software Engineer", "Senior Vice President", "Deputy Managing Director")	Free-form text, no newlines	Professor	Text[p638]
"username"	A username	Free-form text, no newlines	timbl	Username[p638]
"new-password"	A new password (e.g. when creating an account or changing a password)	Free-form text, no newlines	GUMFXbadyrS3	Password[p638]
"current-password"	The current password for the account identified by the username[p636] field (e.g. when logging in)	Free-form text, no newlines	qwerty	Password[p638]
"one-time-code"	One-time code used for verifying user identity	Free-form text, no newlines	123456	Password[p638]
"organization"	Company name corresponding to the person, address, or contact information in the other fields associated with this field	Free-form text, no newlines	World Wide Web Consortium	Text[p638]
"street-address"	Street address (multiple lines, newlines preserved)	Free-form text	32 Vassar Street MIT Room 32-G524	Multiline[p638]
"address-line1"	Street address (one line per field)	Free-form text, no newlines	32 Vassar Street	Text[p638]
"address-line2"		Free-form text, no newlines	MIT Room 32-G524	Text[p638]
"address-line3"		Free-form text, no newlines		Text[p638]

Field name	Meaning	Canonical Format	Canonical Format Example	Control group
"address-level4"	The most fine-grained administrative level[p639], in addresses with four administrative levels	Free-form text, no newlines		Text[p638]
"address-level3"	The third administrative level[p639], in addresses with three or more administrative levels	Free-form text, no newlines		Text[p638]
"address-level2"	The second administrative level[p639], in addresses with two or more administrative levels; in the countries with two administrative levels, this would typically be the city, town, village, or other locality within which the relevant street address is found	Free-form text, no newlines	Cambridge	Text[p638]
"address-level1"	The broadest administrative level[p639] in the address, i.e. the province within which the locality is found; for example, in the US, this would be the state; in Switzerland it would be the canton; in the UK, the post town	Free-form text, no newlines	MA	Text[p638]
"country"	Country code	Valid ISO 3166-1-alpha-2 country code [ISO3166][p1583]	US	Text[p638]
"country-name"	Country name	Free-form text, no newlines; derived from country in some cases[p644]	US	Text[p638]
"postal-code"	Postal code, post code, ZIP code, CEDEX code (if CEDEX, append "CEDEX", and the *arrondissement*, if relevant, to the address-level2[p637] field)	Free-form text, no newlines	02139	Text[p638]
"cc-name"	Full name as given on the payment instrument	Free-form text, no newlines	Tim Berners-Lee	Text[p638]
"cc-given-name"	Given name as given on the payment instrument (in some Western cultures, also known as the *first name*)	Free-form text, no newlines	Tim	Text[p638]
"cc-additional-name"	Additional names given on the payment instrument (in some Western cultures, also known as *middle names*, forenames other than the first name)	Free-form text, no newlines		Text[p638]
"cc-family-name"	Family name given on the payment instrument (in some Western cultures, also known as the *last name* or *surname*)	Free-form text, no newlines	Berners-Lee	Text[p638]
"cc-number"	Code identifying the payment instrument (e.g. the credit card number)	ASCII digits	4114360123456785	Text[p638]
"cc-exp"	Expiration date of the payment instrument	Valid month string[p86]	2014-12	Month[p639]
"cc-exp-month"	Month component of the expiration date of the payment instrument	Valid integer[p80] in the range 1..12	12	Numeric[p639]
"cc-exp-year"	Year component of the expiration date of the payment instrument	Valid integer[p80] greater than zero	2014	Numeric[p639]
"cc-csc"	Security code for the payment instrument (also known as the card security code (CSC), card validation code (CVC), card verification value (CVV), signature panel code (SPC), credit card ID (CCID), etc.)	ASCII digits	419	Text[p638]
"cc-type"	Type of payment instrument	Free-form text, no newlines	Visa	Text[p638]
"transaction-currency"	The currency that the user would prefer the transaction to use	ISO 4217 currency code [ISO4217][p1583]	GBP	Text[p638]
"transaction-amount"	The amount that the user would like for the transaction (e.g. when entering a bid or sale price)	Valid floating-point number[p81]	401.00	Numeric[p639]
"language"	Preferred language	Valid BCP 47 language tag [BCP47][p1579]	en	Text[p638]
"bday"	Birthday	Valid date string[p87]	1955-06-08	Date[p639]
"bday-day"	Day component of birthday	Valid integer[p80] in the range 1..31	8	Numeric[p639]
"bday-month"	Month component of birthday	Valid integer[p80] in the range 1..12	6	Numeric[p639]
"bday-year"	Year component of birthday	Valid integer[p80] greater than zero	1955	Numeric[p639]
"sex"	Gender identity (e.g. Female, Fa'afafine)	Free-form text, no newlines	Male	Text[p638]
"url"	Home page or other web page corresponding to the company, person, address, or contact information in the other fields associated with this field	Valid URL string	https://www.w3.org/People/Berners-Lee/	URL[p638]
"photo"	Photograph, icon, or other image corresponding to the company, person, address, or contact information in the other fields associated with this field	Valid URL string	https://www.w3.org/Press/Stock/Berners-Lee/2001-europaeum-eighth.jpg	URL[p638]
"tel"	Full telephone number, including country code	ASCII digits and U+0020	+1 617 253 5702	Tel[p638]

Field name	Meaning	Canonical Format	Canonical Format Example	Control group
		SPACE characters, prefixed by a U+002B PLUS SIGN character (+)		
"tel-country-code"	Country code component of the telephone number	ASCII digits prefixed by a U+002B PLUS SIGN character (+)	+1	Text[p638]
"tel-national"	Telephone number without the county code component, with a country-internal prefix applied if applicable	ASCII digits and U+0020 SPACE characters	617 253 5702	Text[p638]
"tel-area-code"	Area code component of the telephone number, with a country-internal prefix applied if applicable	ASCII digits	617	Text[p638]
"tel-local"	Telephone number without the country code and area code components	ASCII digits	2535702	Text[p638]
"tel-local-prefix"	First part of the component of the telephone number that follows the area code, when that component is split into two components	ASCII digits	253	Text[p638]
"tel-local-suffix"	Second part of the component of the telephone number that follows the area code, when that component is split into two components	ASCII digits	5702	Text[p638]
"tel-extension"	Telephone number internal extension code	ASCII digits	1000	Text[p638]
"email"	Email address	Valid email address[p549]	timbl@w3.org	Username[p638]
"impp"	URL representing an instant messaging protocol endpoint (for example, "aim:goim?screenname=example" or "xmpp:fred@example.net")	Valid URL string	irc://example.org/timbl,isuser	URL[p638]

The groups correspond to controls as follows:

Text

input[p539] elements with a type[p541] attribute in the Hidden[p546] state
input[p539] elements with a type[p541] attribute in the Text[p546] state
input[p539] elements with a type[p541] attribute in the Search[p546] state
textarea[p606] elements
select[p590] elements

Multiline

input[p539] elements with a type[p541] attribute in the Hidden[p546] state
textarea[p606] elements
select[p590] elements

Password

input[p539] elements with a type[p541] attribute in the Hidden[p546] state
input[p539] elements with a type[p541] attribute in the Text[p546] state
input[p539] elements with a type[p541] attribute in the Search[p546] state
input[p539] elements with a type[p541] attribute in the Password[p550] state
textarea[p606] elements
select[p590] elements

URL

input[p539] elements with a type[p541] attribute in the Hidden[p546] state
input[p539] elements with a type[p541] attribute in the Text[p546] state
input[p539] elements with a type[p541] attribute in the Search[p546] state
input[p539] elements with a type[p541] attribute in the URL[p547] state
textarea[p606] elements
select[p590] elements

Username

input[p539] elements with a type[p541] attribute in the Hidden[p546] state
input[p539] elements with a type[p541] attribute in the Text[p546] state
input[p539] elements with a type[p541] attribute in the Search[p546] state
input[p539] elements with a type[p541] attribute in the Email[p548] state
textarea[p606] elements
select[p590] elements

Tel

input[p539] elements with a type[p541] attribute in the Hidden[p546] state

input[p539] elements with a type[p541] attribute in the Text[p546] state
input[p539] elements with a type[p541] attribute in the Search[p546] state
input[p539] elements with a type[p541] attribute in the Telephone[p547] state
textarea[p606] elements
select[p590] elements

Numeric

input[p539] elements with a type[p541] attribute in the Hidden[p546] state
input[p539] elements with a type[p541] attribute in the Text[p546] state
input[p539] elements with a type[p541] attribute in the Search[p546] state
input[p539] elements with a type[p541] attribute in the Number[p556] state
textarea[p606] elements
select[p590] elements

Month

input[p539] elements with a type[p541] attribute in the Hidden[p546] state
input[p539] elements with a type[p541] attribute in the Text[p546] state
input[p539] elements with a type[p541] attribute in the Search[p546] state
input[p539] elements with a type[p541] attribute in the Month[p552] state
textarea[p606] elements
select[p590] elements

Date

input[p539] elements with a type[p541] attribute in the Hidden[p546] state
input[p539] elements with a type[p541] attribute in the Text[p546] state
input[p539] elements with a type[p541] attribute in the Search[p546] state
input[p539] elements with a type[p541] attribute in the Date[p550] state
textarea[p606] elements
select[p590] elements

Address levels: The "address-level1[p637]" – "address-level4[p637]" 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[p637]" field represents the widest administrative division. Different locales order the fields in different ways; for example, in the US the town (level 2) precedes the state (level 1); while in Japan the prefecture (level 1) precedes the city (level 2) which precedes the district (level 3). Authors are encouraged to provide forms that are presented in a way that matches the country's conventions (hiding, showing, and rearranging fields accordingly as the user changes the country).

4.10.19.7.2 Processing model §[p639]

Each input[p539] element to which the autocomplete[p633] attribute applies[p542], each select[p590] element, and each textarea[p606] element, has an **autofill hint set**, an **autofill scope**, an **autofill field name**, a **non-autofill credential type**, and an **IDL-exposed autofill value**.

The autofill field name[p639] specifies the specific kind of data expected in the field, e.g. "street-address[p636]" or "cc-exp[p637]".

The autofill hint set[p639] identifies what address or contact information type the user agent is to look at, e.g. "shipping[p634] fax[p635]" or "billing[p634]".

The non-autofill credential type[p639] identifies a type of credential that may be offered by the user agent when the user interacts with the field alongside other autofill field[p636] values. If this value is "webauthn" instead of null, selecting a credential of that type will resolve a pending conditional mediation navigator.credentials.get() request, instead of autofilling the field.

Example

For example, a sign-in page could instruct the user agent to either autofill a saved password, or show a public key credential that will resolve a pending navigator.credentials.get() request. A user can select either to sign-in.

```
<input name=password type=password autocomplete="current-password webauthn">
```

The autofill scope[p639] identifies the group of fields whose information concerns the same subject, and consists of the autofill hint set[p639]

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`[p633] 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. Let *field* be the *index*th token in *tokens*.
6. Let *category* be the result of determining a field's category[p641] given *field*.
7. If *category* is null, then jump to the step labeled *default*.
8. If *category* is Off or Automatic, and either *tokens*'s size is greater than 1 or the element's `autocomplete`[p633] attribute is wearing the autofill anchor mantle[p633], then jump to the step labeled *default*.
9. If *category* is Off, set the element's autofill field name[p639] to the string "`off`", set its autofill hint set[p639] to empty, and set its IDL-exposed autofill value[p639] to the string "`off`". Then, return.
10. If *category* is Automatic, set the element's autofill field name[p639] to the string "`on`", set its autofill hint set[p639] to empty, and set its IDL-exposed autofill value[p639] to the string "`on`". Then, return.
11. Let *scope tokens* be an empty list.
12. Let *hint tokens* be an empty set.
13. Let *credential type* be null.
14. Let *IDL value* have the same value as *field*.
15. If *category* is Credential and the *index*th token in *tokens* is an ASCII case-insensitive match for "`webauthn`[p635]", then run the substeps that follow:
    1. Set *credential type* to "`webauthn`".
    2. If the *index*th token in *tokens* is the first entry, then skip to the step labeled *done*.
    3. Decrement *index* by one.
    4. Set *category* to the result of determining a field's category[p641] given the *index*th token in *tokens*.
    5. If *category* is not Normal and *category* is not Contact, then jump to the step labeled *default*.
    6. Set *IDL value* to the concatenation of the *index*th token in *tokens*, a U+0020 SPACE character, and the previous value of *IDL value*.
16. If the *index*th token in *tokens* is the first entry, then skip to the step labeled *done*.
17. Decrement *index* by one.
18. 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:
    - "`home`[p635]"
    - "`work`[p635]"
    - "`mobile`[p635]"
    - "`fax`[p635]"
    - "`pager`[p635]"

    then:
    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*.

5. If the *index*th entry in *tokens* is the first entry, then skip to the step labeled *done*.
6. Decrement *index* by one.

19. If the *index*th token in *tokens* is an ASCII case-insensitive match for one of the strings in the following list:

    - "shipping"
    - "billing"

    then:

    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*.
    5. If the *index*th entry in *tokens* is the first entry, then skip to the step labeled *done*.
    6. Decrement *index* by one.

20. If the *index*th entry in *tokens* is not the first entry, then jump to the step labeled *default*.
21. 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*.
22. Let *section* be the *index*th token in *tokens*, converted to ASCII lowercase.
23. Insert *section* at the start of *scope tokens*.
24. Let *IDL value* be the concatenation of *section*, a U+0020 SPACE character, and the previous value of *IDL value*.
25. *Done*: Set the element's autofill hint set to *hint tokens*.
26. Set the element's non-autofill credential type to *credential type*.
27. Set the element's autofill scope to *scope tokens*.
28. Set the element's autofill field name to *field*.
29. Set the element's IDL-exposed autofill value to *IDL value*.
30. Return.
31. *Default*: Set the element's IDL-exposed autofill value to the empty string, and its autofill hint set and autofill scope to empty.
32. If the element's autocomplete attribute is wearing the autofill anchor mantle, then set the element's autofill field name to the empty string and return.
33. Let *form* be the element's form owner, if any, or null otherwise.
34. If *form* is not null and *form*'s autocomplete attribute is in the Off state, then set the element's autofill field name to "off".

    Otherwise, set the element's autofill field name to "on".

To **determine a field's category**, given *field*:

1. If the *field* is not an ASCII case-insensitive match for one of the tokens given in the first column of the following table, then return null.

Token	Category
"off"	Off
"on"	Automatic
"name"	Normal
"honorific-prefix"	Normal
"given-name"	Normal

Token	Category
"additional-name[p636]"	Normal
"family-name[p636]"	Normal
"honorific-suffix[p636]"	Normal
"nickname[p636]"	Normal
"organization-title[p636]"	Normal
"username[p636]"	Normal
"new-password[p636]"	Normal
"current-password[p636]"	Normal
"one-time-code[p636]"	Normal
"organization[p636]"	Normal
"street-address[p636]"	Normal
"address-line1[p636]"	Normal
"address-line2[p636]"	Normal
"address-line3[p636]"	Normal
"address-level4[p637]"	Normal
"address-level3[p637]"	Normal
"address-level2[p637]"	Normal
"address-level1[p637]"	Normal
"country[p637]"	Normal
"country-name[p637]"	Normal
"postal-code[p637]"	Normal
"cc-name[p637]"	Normal
"cc-given-name[p637]"	Normal
"cc-additional-name[p637]"	Normal
"cc-family-name[p637]"	Normal
"cc-number[p637]"	Normal
"cc-exp[p637]"	Normal
"cc-exp-month[p637]"	Normal
"cc-exp-year[p637]"	Normal
"cc-csc[p637]"	Normal
"cc-type[p637]"	Normal
"transaction-currency[p637]"	Normal
"transaction-amount[p637]"	Normal
"language[p637]"	Normal
"bday[p637]"	Normal
"bday-day[p637]"	Normal
"bday-month[p637]"	Normal
"bday-year[p637]"	Normal
"sex[p637]"	Normal
"url[p637]"	Normal
"photo[p637]"	Normal
"tel[p637]"	Contact
"tel-country-code[p638]"	Contact
"tel-national[p638]"	Contact
"tel-area-code[p638]"	Contact
"tel-local[p638]"	Contact
"tel-local-prefix[p638]"	Contact
"tel-local-suffix[p638]"	Contact
"tel-extension[p638]"	Contact
"email[p638]"	Contact
"impp[p638]"	Contact
"webauthn[p635]"	Credential

2. Return the value of the cell in the second column of that row.

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.

Note

In addition, when an element's autofill field name is "`off`", values are reset when reactivating a document.

Example

Banks frequently do not want UAs to prefill login information:

```
<p><label>Account: <input type="text" name="ac" autocomplete="off"></label></p>
<p><label>PIN: <input type="password" name="pin" autocomplete="off"></label></p>
```

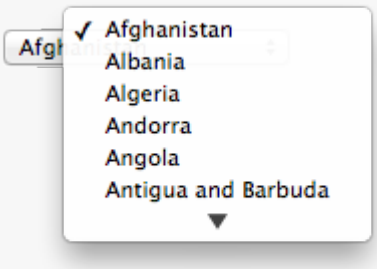
When an element's autofill field name is *not* "`off`", the user agent may store the control's data, and may offer previously stored values to the user.

Example

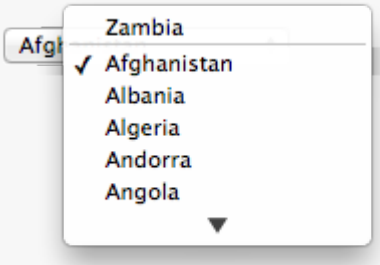
For example, suppose a user visits a page with this control:

```
<select name="country">
 <option>Afghanistan
 <option>Albania
 <option>Algeria
 <option>Andorra
 <option>Angola
 <option>Antigua and Barbuda
 <option>Argentina
 <option>Armenia
 <!-- ... -->
 <option>Yemen
 <option>Zambia
 <option>Zimbabwe
</select>
```

This might render as follows:



Suppose that on the first visit to this page, the user selects "Zambia". On the second visit, the user agent could duplicate the entry for Zambia at the top of the list, so that the interface instead looks like this:



When the autofill field name[p639] is "on[p635]", 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[p628] value, the position of the element in its tree, what other fields exist in the form, and so forth.

When the autofill field name[p639] is one of the names of the autofill fields[p636] 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[p639] should be used to select amongst multiple possible suggestions.

Example

For example, if a user once entered one address into fields that used the "shipping[p634]" keyword, and another address into fields that used the "billing[p634]" keyword, then in subsequent forms only the first address would be suggested for form controls whose autofill hint set[p639] contains the keyword "shipping[p634]". Both addresses might be suggested, however, for address-related form controls whose autofill hint set[p639] does not contain either keyword.

↪ **When wearing the autofill anchor mantle[p633]...**

When the autofill field name[p639] is not the empty string, then the user agent must act as if the user had specified the control's data[p643] for the given autofill hint set[p639], autofill scope[p639], and autofill field name[p639] combination.

When the user agent **autofills form controls**, elements with the same form owner[p626] and the same autofill scope[p639] must use data relating to the same person, address, payment instrument, and contact details. When a user agent autofills "country[p637]" and "country-name[p637]" fields with the same form owner[p626] and autofill scope[p639], and the user agent has a value for the country[p637]" field(s), then the "country-name[p637]" 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[p639], form owner[p626], and autofill scope[p639] 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[p637]" field and a "cc-exp-month[p637]" 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[p633] also. Consider the following markup snippet:

```
<form>
 <input type=hidden autocomplete="nickname" value="TreePlate">
 <input type=text autocomplete="nickname">
</form>
```

The only value that a conforming user agent could suggest in the text control is "TreePlate", the value given by the hidden input[p539] element.

The "section-*" tokens in the autofill scope[p639] 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[p643], and must be done at a time where the element is mutable[p626] (e.g. just after the element has been inserted into the document, or when the user agent stops parsing[p1458]). User agents must only prefill controls using values that the user could have entered.

Example

For example, if a select[p590] element only has option[p601] elements with values "Steve" and "Rebecca", "Jay", and "Bob", and has an autofill field name[p639] "given-name[p636]", 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[p590] element to the value "Evan", since the user could not have done so themselves.

A user agent prefilling a form control must not discriminate between form controls that are in a document tree and those that are connected; that is, it is not conforming to make the decision on whether or not to autofill based on whether the element's root is a shadow root versus a Document[p135].

A user agent prefilling a form control's value[p626] must not cause that control to suffer from a type mismatch[p651], suffer from being too long[p652], suffer from being too short[p652], suffer from an underflow[p652], suffer from an overflow[p652], or suffer from a step mismatch[p652]. A user agent prefilling a form control's value[p626] must not cause that control to suffer from a pattern mismatch[p651] either. Where possible given the control's constraints, user agents must use the format given as canonical in the aforementioned table. Where it's not possible for the canonical format to be used, user agents should use heuristics to attempt to convert values so that they can be used.

Example

For example, if the user agent knows that the user's middle name is "Ines", and attempts to prefill a form control that looks like this:

```
<input name=middle-initial maxlength=1 autocomplete="additional-name">
```

...then the user agent could convert "Ines" to "I" and prefill it that way.

Example

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:

`<input name=b type=month autocomplete="bday">`	2012-07	The day is dropped since the Month[p552] state only accepts a month/year combination. (Note that this example is non-conforming, because the autofill field name[p639] bday[p637] is not allowed with the Month[p552] state.)

`<select name=c autocomplete="bday"> <option>Jan <option>Feb ... <option>Jul <option>Aug ... </select>`	July	The user agent picks the month from the listed options, either by noticing there are twelve options and picking the 7th, or by recognizing that one of the strings (three characters "Jul" followed by a newline and a space) is a close match for the name of the month (July) in one of the user agent's supported languages, or through some other similar mechanism.
`<input name=a type=number min=1 max=12 autocomplete="bday-month">`	7	User agent converts "July" to a month number in the range 1..12, like the field.
`<input name=a type=number min=0 max=11 autocomplete="bday-month">`	6	User agent converts "July" to a month number in the range 0..11, like the field.
`<input name=a type=number min=1 max=11 autocomplete="bday-month">`		User agent doesn't fill in the field, since it can't make a good guess as to what the form expects.

A user agent may allow the user to override an element's autofill field name[p639], e.g. to change it from "off[p635]" to "on[p635]" to allow values to be remembered and prefilled despite the page author's objections, or to always "off[p635]", never remembering values.

More specifically, user agents may in particular consider replacing the autofill field name[p639] of form controls that match the description given in the first column of the following table, when their autofill field name[p639] is either "on[p635]" or "off[p635]", 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[p639] 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[p532] that share the same form owner[p626].

Form control	New autofill field name[p639]
an input[p539] element whose type[p541] attribute is in the Text[p546] state that is followed by an input[p539] element whose type[p541] attribute is in the Password[p550] state	"username[p636]"
an input[p539] element whose type[p541] attribute is in the Password[p550] state that is preceded by an input[p539] element whose autofill field name[p639] is "username[p636]"	"current-password[p636]"
an input[p539] element whose type[p541] attribute is in the Password[p550] state that is preceded by an input[p539] element whose autofill field name[p639] is "current-password[p636]"	"new-password[p636]"
an input[p539] element whose type[p541] attribute is in the Password[p550] state that is preceded by an input[p539] element whose autofill field name[p639] is "new-password[p636]"	"new-password[p636]"

The **autocomplete** IDL attribute, on getting, must return the element's IDL-exposed autofill value[p639].

4.10.20 APIs for the text control selections §[p646]

The input[p539] and textarea[p606] elements define several attributes and methods for handling their selection. Their shared algorithms are defined here.

For web developers (non-normative)

***element*.select[p648]()**

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:

"select"
: Selects the newly inserted text.

"start"
: Moves the selection to just before the inserted text.

"end"
: Moves the selection to just after the selected text.

"preserve"
: Attempts to preserve the selection. This is the default.

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 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**.

Example

The use of API value instead of raw value for textarea elements means that U+000D (CR) characters are normalized away. For example,

```
<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.

Whenever the relevant value changes for an element to which these APIs apply, run these steps:

1. If the element has a selection[p647]:

    1. If the start of the selection is now past the end of the relevant value[p647], set it to the end of the relevant value[p647].

    2. If the end of the selection is now past the end of the relevant value[p647], set it to the end of the relevant value[p647].

    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[p647], then instead set the element's text entry cursor position[p647] to the end of the relevant value[p647], removing any selection.

2. Otherwise, the element must have a text entry cursor position[p647] position. If it is now past the end of the relevant value[p647], set it to the end of the relevant value[p647].

Note

In some cases where the relevant value[p647] changes, other parts of the specification will also modify the text entry cursor position[p647], beyond just the clamping steps above. For example, see the `value`[p610] setter for `textarea`[p606].

Where possible, user interface features for changing the text selection[p647] in `input`[p539] and `textarea`[p606] elements must be implemented using the set the selection range[p649] algorithm so that, e.g., all the same events fire.

The selections[p647] of `input`[p539] and `textarea`[p606] 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[p648] 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[p648] 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 direction[p648] 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:

✔ MDN

1. If this element is an `input`[p539] element, and either `select()`[p648] does not apply[p542] to this element or the corresponding control has no selectable text, return.

    Example

    For instance, in a user agent where `<input type=color>`[p559] 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[p649] with 0 and infinity.

The **`selectionStart`** attribute's getter must run the following steps:

1. If this element is an `input`[p539] element, and `selectionStart`[p648] does not apply[p542] to this element, return null.

2. If there is no selection[p647], return the code unit offset within the relevant value[p647] to the character that immediately follows the text entry cursor[p647].

3. Return the code unit offset within the relevant value[p647] to the character that immediately follows the start of the selection[p647].

The `selectionStart`[p648] attribute's setter must run the following steps:

1. If this element is an `input`[p539] element, and `selectionStart`[p648] does not apply[p542] to this element, throw an "`InvalidStateError`" `DOMException`.

2. Let *end* be the value of this element's selectionEnd[p649] attribute.
3. If *end* is less than the given value, set *end* to the given value.
4. Set the selection range[p649] with the given value, *end*, and the value of this element's selectionDirection[p649] attribute.

The **selectionEnd** attribute's getter must run the following steps:

1. If this element is an input[p539] element, and selectionEnd[p649] does not apply[p542] to this element, return null.
2. If there is no selection[p647], return the code unit offset within the relevant value[p647] to the character that immediately follows the text entry cursor[p647].
3. Return the code unit offset within the relevant value[p647] to the character that immediately follows the end of the selection[p647].

The selectionEnd[p649] attribute's setter must run the following steps:

1. If this element is an input[p539] element, and selectionEnd[p649] does not apply[p542] to this element, throw an "InvalidStateError" DOMException.
2. Set the selection range[p649] with the value of this element's selectionStart[p648] attribute, the given value, and the value of this element's selectionDirection[p649] attribute.

The **selectionDirection** attribute's getter must run the following steps:

1. If this element is an input[p539] element, and selectionDirection[p649] does not apply[p542] to this element, return null.
2. Return this element's selection direction[p648].

The selectionDirection[p649] attribute's setter must run the following steps:

1. If this element is an input[p539] element, and selectionDirection[p649] does not apply[p542] to this element, throw an "InvalidStateError" DOMException.
2. Set the selection range[p649] with the value of this element's selectionStart[p648] attribute, the value of this element's selectionEnd[p649] 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[p539] element, and setSelectionRange()[p649] does not apply[p542] to this element, throw an "InvalidStateError" DOMException.
2. Set the selection range[p649] 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 0.
2. If *end* is null, let *end* be 0.
3. Set the selection[p647] of the text control to the sequence of code units within the relevant value[p647] starting with the code unit at the *start*th 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[p647] 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[p648] of the text control to *direction*.
6. If the previous steps caused the selection[p647] of the text control to be modified (in either extent or direction[p648]), then queue an element task[p1195] on the user interaction task source[p1204] given the element to fire an event named select[p1576] 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[p539] element, and setRangeText()[p650] does not apply[p542] to this element, throw an "InvalidStateError" DOMException.

2. Set this element's dirty value flag[p626] to true.

3. If the method has only one argument, then let *start* and *end* have the values of the selectionStart[p648] attribute and the selectionEnd[p649] 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[p647] of the text control, then set it to the length of the relevant value[p647] of the text control.

6. If *end* is greater than the length of the relevant value[p647] of the text control, then set it to the length of the relevant value[p647] of the text control.

7. Let *selection start* be the current value of the selectionStart[p648] attribute.

8. Let *selection end* be the current value of the selectionEnd[p649] attribute.

9. If *start* is less than *end*, delete the sequence of code units within the element's relevant value[p647] starting with the code unit at the *start*th 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[p647] of the text control, immediately before the *start*th 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[p649] with *selection start* and *selection end*.

The setRangeText()[p650] method uses the following enumeration:

IDL

```
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[p539] or textarea[p606] element.

Example

To add some text at the start of a text control, while maintaining the text selection, the three attributes must be preserved:

```
var oldStart = control.selectionStart;
var oldEnd = control.selectionEnd;
var oldDirection = control.selectionDirection;
var prefix = "http://";
control.value = prefix + control.value;
control.setSelectionRange(oldStart + prefix.length, oldEnd + prefix.length, oldDirection);
```

...where *control* is the input[p539] or textarea[p606] element.

4.10.21 Constraints §[p65]

4.10.21.1 Definitions §[p65]

A submittable element[p532] 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[p651] if it has a datalist[p598] element ancestor.)

An element can have a **custom validity error message** defined. Initially, an element must have its custom validity error message[p651] set to the empty string. When its value is not the empty string, the element is suffering from a custom error[p652]. It can be set using the setCustomValidity()[p654] method, except for form-associated custom elements[p797]. Form-associated custom elements[p797] can have a custom validity error message[p651] set via their ElementInternals[p809] object's setValidity()[p812] method. The user agent should use the custom validity error message[p651] 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[p626] but has a `required` attribute (input[p539] required[p571], textarea[p606] required[p610]); or, more complicated rules for select[p590] elements and controls in radio button groups[p562], as specified in their sections.

When the setValidity()[p812] method sets `valueMissing` flag to true for a form-associated custom element[p797].

Suffering from a type mismatch

When a control that allows arbitrary user input has a value[p626] that is not in the correct syntax (Email[p548], URL[p547]).

When the setValidity()[p812] method sets `typeMismatch` flag to true for a form-associated custom element[p797].

Suffering from a pattern mismatch

When a control has a value[p626] that doesn't satisfy the pattern[p573] attribute.

When the setValidity()[p812] method sets patternMismatch flag to true for a form-associated custom element[p797].

Suffering from being too long

When a control has a value[p626] that is too long for the form control maxlength attribute[p629] (input[p539] maxlength[p569], textarea[p606] maxlength[p609]).

When the setValidity()[p812] method sets tooLong flag to true for a form-associated custom element[p797].

Suffering from being too short

When a control has a value[p626] that is too short for the form control minlength attribute[p630] (input[p539] minlength[p569], textarea[p606] minlength[p609]).

When the setValidity()[p812] method sets tooShort flag to true for a form-associated custom element[p797].

Suffering from an underflow

When a control has a value[p626] that is not the empty string and is too low for the min[p574] attribute.

When the setValidity()[p812] method sets rangeUnderflow flag to true for a form-associated custom element[p797].

Suffering from an overflow

When a control has a value[p626] that is not the empty string and is too high for the max[p574] attribute.

When the setValidity()[p812] method sets rangeOverflow flag to true for a form-associated custom element[p797].

Suffering from a step mismatch

When a control has a value[p626] that doesn't fit the rules given by the step[p575] attribute.

When the setValidity()[p812] method sets stepMismatch flag to true for a form-associated custom element[p797].

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()[p812] method sets badInput flag to true for a form-associated custom element[p797].

Suffering from a custom error

When a control's custom validity error message[p651] (as set by the element's setCustomValidity()[p654] method or ElementInternals[p809]'s setValidity()[p812] method) is not the empty string.

Note

An element can still suffer from these states even when the element is disabled[p630]; 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[p651].

4.10.21.2 Constraint validation [§ p652]

When the user agent is required to **statically validate the constraints** of form[p532] 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[p532] whose form owner[p626] 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[p651], then move on to the next element.
    2. Otherwise, if *field* satisfies its constraints[p652], 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`[p1575] 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`[p532] element *form*, then the user agent must run the following steps:

1. Statically validate the constraints[p652] 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[p881] 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[p797], user agents should use their validation anchor[p812] 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[p562] are marked as required, only one error need be reported).
    - If one of the controls is not being rendered[p1488] (e.g. it has the `hidden`[p862] attribute set), then user agents may report a script error.
4. Return a *negative* result.

4.10.21.3 The constraint validation API §[p653]

For web developers (non-normative)

***element*.willValidate**[p654]

Returns true if the element will be validated when the form is submitted; false otherwise.

***element*.setCustomValidity**[p654]**(*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**[p654]**.valueMissing**[p655]

Returns true if the element has no value but is a required field; false otherwise.

***element*.validity**[p654]**.typeMismatch**[p655]

Returns true if the element's value is not in the correct syntax; false otherwise.

***element*.validity**[p654]**.patternMismatch**[p655]

Returns true if the element's value doesn't match the provided pattern; false otherwise.

***element*.validity**[p654]**.tooLong**[p655]

Returns true if the element's value is longer than the provided maximum length; false otherwise.

***element*.validity**[p654]**.tooShort**[p655]

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[p654].rangeUnderflow[p655]**

Returns true if the element's value is lower than the provided minimum; false otherwise.

***element*.validity[p654].rangeOverflow[p655]**

Returns true if the element's value is higher than the provided maximum; false otherwise.

***element*.validity[p654].stepMismatch[p655]**

Returns true if the element's value doesn't fit the rules given by the step[p575] attribute; false otherwise.

***element*.validity[p654].badInput[p655]**

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[p654].customError[p655]**

Returns true if the element has a custom error; false otherwise.

***element*.validity[p654].valid[p655]**

Returns true if the element's value has no validity problems; false otherwise.

***valid* = *element*.checkValidity[p656]()**

Returns true if the element's value has no validity problems; false otherwise. Fires an invalid[p1575] event at the element in the latter case.

***valid* = *element*.reportValidity[p656]()**

Returns true if the element's value has no validity problems; otherwise, returns false, fires an invalid[p1575] event at the element, and (if the event isn't canceled) reports the problem to the user.

***element*.validationMessage[p656]**

Returns the error message that would be shown to the user if the element was to be checked for validity.

The **willValidate** attribute's getter must return true, if this element is a candidate for constraint validation[p651], and false otherwise (i.e., false if any conditions are barring it from constraint validation[p651]).

The **willValidate** attribute of ElementInternals[p809] interface, on getting, must throw a "NotSupportedError" DOMException if the target element[p810] is not a form-associated custom element[p797]. Otherwise, it must return true if the target element[p810] is a candidate for constraint validation[p651], and false otherwise.

The **setCustomValidity(*error*)** method steps are:

1. Set *error* to the result of normalizing newlines given *error*.
2. Set the custom validity error message[p651] to *error*.

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()[p654] method to set an appropriate message.

```
<label>Feeling: <input name=f type="text" oninput="check(this)"></label>
<script>
 function check(input) {
   if (input.value == "good" ||
       input.value == "fine" ||
       input.value == "tired") {
     input.setCustomValidity('"' + input.value + '" is not a feeling.');
   } else {
     // input is fine -- reset the error message
     input.setCustomValidity('');
   }
 }
</script>
```

The **validity** attribute's getter must return a ValidityState[p655] object that represents the validity states[p651] of this element. This

object is live[p48].

The **validity** attribute of ElementInternals[p809] interface, on getting, must throw a "NotSupportedError" DOMException if the target element[p810] is not a form-associated custom element[p797]. Otherwise, it must return a ValidityState[p655] object that represents the validity states[p651] of the target element[p810]. This object is live[p48].

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[p655] object has the following attributes. On getting, they must return true if the corresponding condition given in the following list is true, and false otherwise.

valueMissing
: The control is suffering from being missing[p651].

typeMismatch
: The control is suffering from a type mismatch[p651].

patternMismatch
: The control is suffering from a pattern mismatch[p651].

tooLong
: The control is suffering from being too long[p652].

tooShort
: The control is suffering from being too short[p652].

rangeUnderflow
: The control is suffering from an underflow[p652].

rangeOverflow
: The control is suffering from an overflow[p652].

stepMismatch
: The control is suffering from a step mismatch[p652].

badInput
: The control is suffering from bad input[p652].

customError
: The control is suffering from a custom error[p652].

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[p651] and does not satisfy its constraints[p652]:
    1. Fire an event named invalid[p1575] 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[p655] on this element.

The **checkValidity()** method of the ElementInternals[p809] interface must run these steps:

1. Let *element* be this ElementInternals[p809]'s target element[p810].
2. If *element* is not a form-associated custom element[p797], then throw a "NotSupportedError" DOMException.
3. Run the check validity steps[p655] on *element*.

The **report validity steps** for an element *element* are:

1. If *element* is a candidate for constraint validation[p651] and does not satisfy its constraints[p652]:
    1. Let *report* be the result of firing an event named invalid[p1575] 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[p881] 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.
    3. Return false.
2. Return true.

The **reportValidity()** method, when invoked, must run the report validity steps[p656] on this element.

The **reportValidity()** method of the ElementInternals[p809] interface must run these steps:

1. Let *element* be this ElementInternals[p809]'s target element[p810].
2. If *element* is not a form-associated custom element[p797], then throw a "NotSupportedError" DOMException.
3. Run the report validity steps[p656] on *element*.

The **validationMessage** attribute's getter must run these steps:

1. If this element is not a candidate for constraint validation[p651] or if this element satisfies its constraints[p652], 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[p651] and is suffering from a custom error[p652], then the custom validity error message[p651] should be present in the return value.

4.10.21.4 Security §[p656]

Servers should not rely on client-side validation. Client-side validation can be intentionally bypassed by hostile users, and unintentionally bypassed by users of older user agents or automated tools that do not implement these features. The constraint validation features are only intended to improve the user experience, not to provide any kind of security mechanism.

4.10.22 Form submission §[p657]

4.10.22.1 Introduction §[p657]

This section is non-normative.

When a form is submitted, the data in the form is converted into the structure specified by the enctype[p632], and then sent to the destination specified by the action[p631] using the given method[p631].

For example, take the following form:

```
<form action="/find.cgi" method=get>
 <input type=text name=t>
 <input type=search name=q>
 <input type=submit>
</form>
```

If the user types in "cats" in the first field and "fur" in the second, and then hits the submit button, then the user agent will load `/find.cgi?t=cats&q=fur`.

On the other hand, consider this form:

```
<form action="/find.cgi" method=post enctype="multipart/form-data">
 <input type=text name=t>
 <input type=search name=q>
 <input type=submit>
</form>
```

Given the same user input, the result on submission is quite different: the user agent instead does an HTTP POST to the given URL, with as the entity body something like the following text:

```
------kYFrd4jNJEgCervE
Content-Disposition: form-data; name="t"

cats
------kYFrd4jNJEgCervE
Content-Disposition: form-data; name="q"

fur
------kYFrd4jNJEgCervE--
```

4.10.22.2 Implicit submission §[p657]

A `form`[p532] element's **default button** is the first submit button[p532] in tree order whose form owner[p626] is that `form`[p532] 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[p875] implicitly submits the form), then doing so for a form, whose default button[p657] has activation behavior and is not disabled[p630], must cause the user agent to fire a `click` event[p1218] at that default button[p657].

Note

There are pages on the web that are only usable if there is a way to implicitly submit forms, so user agents are strongly encouraged to support this.

If the form has no submit button[p532], then the implicit submission mechanism must perform the following steps:

1. If the form has more than one field that blocks implicit submission[p657], then return.
2. Submit[p658] the `form`[p532] element from the `form`[p532] element itself with *userInvolvement*[p658] set to "`activation`[p1062]".

For the purpose of the previous paragraph, an element is a **field that blocks implicit submission** of a `form`[p532] element if it is an `input`[p539] element whose form owner[p626] is that `form`[p532] element and whose `type`[p541] attribute is in one of the following states: Text[p546],

Search[p546], Telephone[p547], URL[p547], Email[p548], Password[p550], Date[p550], Month[p552], Week[p553], Time[p554], Local Date and Time[p554], Number[p556]

4.10.22.3 Form submission algorithm §[p658]

Each `form`[p532] element has a **constructing entry list** boolean, initially false.

Each `form`[p532] element has a **firing submission events** boolean, initially false.

To **submit** a `form`[p532] element *form* from an element *submitter* (typically a button), given an optional boolean ***submitted from `submit()` method*** (default false) and an optional user navigation involvement[p1062] ***userInvolvement*** (default "`none`[p1062]"):

1. If *form* cannot navigate[p321], then return.
2. If *form*'s constructing entry list[p658] is true, then return.
3. Let *form document* be *form*'s node document.
4. If *form document*'s active sandboxing flag set[p959] has its sandboxed forms browsing context flag[p957] set, then return.
5. If *submitted from* `submit()`[p535] *method* is false:
    1. If *form*'s firing submission events[p658] is true, then return.
    2. Set *form*'s firing submission events[p658] to true.
    3. For each element *field* in the list of submittable elements[p532] whose form owner[p626] is *form*, set *field*'s user validity[p626] to true.
    4. If the *submitter* element's no-validate state[p632] is false, then interactively validate the constraints[p653] 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):
        1. Set *form*'s firing submission events[p658] to false.
        2. Return.
    5. Let *submitterButton* be null if *submitter* is *form*. Otherwise, let *submitterButton* be *submitter*.
    6. Let *shouldContinue* be the result of firing an event named `submit`[p1576] at *form* using `SubmitEvent`[p665], with the `submitter`[p665] attribute initialized to *submitterButton*, the `bubbles` attribute initialized to true, and the `cancelable` attribute initialized to true.
    7. Set *form*'s firing submission events[p658] to false.
    8. If *shouldContinue* is false, then return.
    9. If *form* cannot navigate[p321], then return.

    > **Note**
    > *Cannot navigate[p321] is run again as dispatching the `submit`[p1576] event could have changed the outcome.*
6. Let *encoding* be the result of picking an encoding for the form[p663].
7. Let *entry list* be the result of constructing the entry list[p661] with *form*, *submitter*, and *encoding*.
8. Assert: *entry list* is not null.
9. If *form* cannot navigate[p321], then return.

    > **Note**
    > *Cannot navigate[p321] is run again as dispatching the `formdata`[p1575] event in constructing the entry list[p661] could have changed the outcome.*
10. Let *method* be the *submitter* element's method[p631].
11. If *method* is dialog[p631]:

1. If *form* does not have an ancestor `dialog` element, then return.
2. Let *subject* be *form*'s nearest ancestor `dialog` element.
3. Let *result* be null.
4. If *submitter* is an `input` element whose `type` attribute is in the Image Button state:
    1. Let (*x*, *y*) be the selected coordinate.
    2. Set *result* to the concatenation of *x*, ",", and *y*.
5. Otherwise, if *submitter* is a submit button, then set *result* to *submitter*'s optional value.
6. Close the dialog *subject* with *result* and null.
7. 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. Let *parsed action* be the result of encoding-parsing a URL given *action*, relative to *submitter*'s node document.
15. If *parsed action* is failure, then return.
16. Let *scheme* be the scheme of *parsed action*.
17. Let *enctype* be the *submitter* element's enctype.
18. Let *formTarget* be null.
19. If the *submitter* element is a submit button and it has a `formtarget` attribute, then set *formTarget* to the `formtarget` attribute value.
20. Let *target* be the result of getting an element's target given *submitter*'s form owner and *formTarget*.
21. Let *noopener* be the result of getting an element's noopener with *form*, *parsed action*, and *target*.
22. Let *targetNavigable* be the first return value of applying the rules for choosing a navigable given *target*, *form*'s node navigable, and *noopener*.
23. If *targetNavigable* is null, then return.
24. Let *historyHandling* be "`auto`".
25. If *form document* equals *targetNavigable*'s active document, and *form document* has not yet completely loaded, then set *historyHandling* to "`replace`".
26. Select the appropriate row in the table below based on *scheme* as given by the first cell of each row. Then, select the appropriate cell on that row based on *method* as given in the first cell of each column. Then, jump to the steps named in that cell and defined below the table.

	GET	**POST**
`http`	Mutate action URL	Submit as entity body
`https`	Mutate action URL	Submit as entity body
`ftp`	Get action URL	Get action URL
`javascript`	Get action URL	Get action URL
`data`	Mutate action URL	Get action URL
`mailto`	Mail with headers	Mail as body

If *scheme* is not one of those listed in this table, then the behavior is not defined by this specification. User agents should, in the absence of another specification defining this, act in a manner analogous to that defined in this specification for similar schemes.

Each `form` element has a **planned navigation**, which is either null or a task; when the `form` is first created, its planned navigation must be set to null. In the behaviors described below, when the user agent is required to **plan to navigate** to a URL *url* given an optional POST resource-or-null *postResource* (default null), it must run the following steps:

1. Let *referrerPolicy* be the empty string.
2. If the `form`[p532] element's link types[p326] include the `noreferrer`[p338] keyword, then set *referrerPolicy* to "`no-referrer`".
3. If the `form`[p532] has a non-null planned navigation[p659], remove it from its task queue[p1193].
4. Queue an element task[p1195] on the DOM manipulation task source[p1203] given the `form`[p532] element and the following steps:
    1. Set the `form`[p532]'s planned navigation[p659] to null.
    2. Navigate[p1063] *targetNavigable* to *url* using the `form`[p532] element's node document, with *historyHandling*[p1063] set to *historyHandling*, *userInvolvement*[p1063] set to *userInvolvement*, *sourceElement*[p1063] set to *submitter*, *referrerPolicy*[p1063] set to *referrerPolicy*, *documentResource*[p1063] set to *postResource*, and *formDataEntryList*[p1063] set to *entry list*.
5. Set the `form`[p532]'s planned navigation[p659] to the just-queued task[p1194].

The behaviors are as follows:

Mutate action URL

Let *pairs* be the result of converting to a list of name-value pairs[p663] with *entry list*.

Let *query* be the result of running the `application/x-www-form-urlencoded` serializer with *pairs* and *encoding*.

Set *parsed action*'s query component to *query*.

Plan to navigate[p659] to *parsed action*.

Submit as entity body

Assert: *method* is POST[p631].

Switch on *enctype*:

↪ **`application/x-www-form-urlencoded`**[p632]

Let *pairs* be the result of converting to a list of name-value pairs[p663] with *entry list*.

Let *body* be the result of running the `application/x-www-form-urlencoded` serializer with *pairs* and *encoding*.

Set *body* to the result of encoding *body*.

Let *mimeType* be "`application/x-www-form-urlencoded`".

↪ **`multipart/form-data`**[p632]

Let *body* be the result of running the `multipart/form-data` encoding algorithm[p664] with *entry list* and *encoding*.

Let *mimeType* be the isomorphic encoding of the concatenation of "`multipart/form-data; boundary=`" and the `multipart/form-data` boundary string[p664] generated by the `multipart/form-data` encoding algorithm[p664].

↪ **`text/plain`**[p632]

Let *pairs* be the result of converting to a list of name-value pairs[p663] with *entry list*.

Let *body* be the result of running the `text/plain` encoding algorithm[p664] with *pairs*.

Set *body* to the result of encoding *body* using *encoding*.

Let *mimeType* be "`text/plain`".

Plan to navigate[p659] to *parsed action* given a POST resource[p1055] whose request body[p1055] is *body* and request content-type[p1055] is *mimeType*.

Get action URL

Plan to navigate[p659] to *parsed action*.

Note

entry list *is discarded.*

Mail with headers

Let *pairs* be the result of converting to a list of name-value pairs[p663] with *entry list*.

Let *headers* be the result of running the application/x-www-form-urlencoded serializer with *pairs* 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[p659] to *parsed action*.

Mail as body

Let *pairs* be the result of converting to a list of name-value pairs[p663] with *entry list*.

Switch on *enctype*:

↪ **text/plain**[p632]

Let *body* be the result of running the text/plain encoding algorithm[p664] with *pairs*.

Set *body* to the result of running UTF-8 percent-encode on *body* using the default encode set. [URL][p1587]

↪ **Otherwise**

Let *body* be the result of running the application/x-www-form-urlencoded serializer with *pairs* 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[p659] to *parsed action*.

4.10.22.4 Constructing the entry list §[p66 1]

An **entry list** is a list of entries[p661], typically representing the contents of a form. An **entry** is a tuple consisting of a **name** (a scalar value string) and a **value** (either a scalar value string or a File object).

To **create an entry** given a string *name*, a string or Blob object *value*, and optionally a scalar value string *filename*:

1. Set *name* to the result of converting *name* into a scalar value string.
2. If *value* is a string, then set *value* to the result of converting *value* into a scalar value string.
3. Otherwise:
    1. If *value* is not a File object, then set *value* to a new File object, representing the same bytes, whose name attribute value is "blob".
    2. If *filename* is given, then set *value* to a new File object, representing the same bytes, whose name attribute is *filename*.

Note

These operations will create a new File object if either filename *is given or the passed Blob is not a File object. In those cases, the identity of the passed Blob object is not kept.*

4. Return an entry[p661] whose name[p661] is *name* and whose value[p661] is *value*.

To **construct the entry list** given a *form*, an optional *submitter* (default null), and an optional *encoding* (default UTF-8):

1. If *form*'s constructing entry list[p658] is true, then return null.
2. Set *form*'s constructing entry list[p658] to true.
3. Let *controls* be a list of all the submittable elements[p532] whose form owner[p626] is *form*, in tree order.
4. Let *entry list* be a new empty entry list[p661].
5. For each element *field* in *controls*, in tree order:
    1. If any of the following are true:
        - *field* has a `datalist`[p598] element ancestor;
        - *field* is disabled[p630];
        - *field* is a button[p532] but it is not *submitter*;
        - *field* is an `input`[p539] element whose `type`[p541] attribute is in the Checkbox[p561] state and whose checkedness[p626] is false; or
        - *field* is an `input`[p539] element whose `type`[p541] attribute is in the Radio Button[p561] state and whose checkedness[p626] is false,

        then continue.
    2. If the *field* element is an `input`[p539] element whose `type`[p541] attribute is in the Image Button[p566] state:
        1. If the *field* element is not *submitter*, then continue.
        2. If the *field* element has a `name`[p628] attribute specified and its value is not the empty string, let *name* be that value followed by U+002E (.). Otherwise, let *name* be the empty string.
        3. Let $name_x$ be the concatenation of *name* and U+0078 (x).
        4. Let $name_y$ be the concatenation of *name* and U+0079 (y).
        5. Let (x, y) be the selected coordinate[p567].
        6. Create an entry[p661] with $name_x$ and x, and append it to *entry list*.
        7. Create an entry[p661] with $name_y$ and y, and append it to *entry list*.
        8. Continue.
    3. If the *field* is a form-associated custom element[p797], then perform the entry construction algorithm[p812] given *field* and *entry list*, then continue.
    4. If either the *field* element does not have a `name`[p628] attribute specified, or its `name`[p628] attribute's value is the empty string, then continue.
    5. Let *name* be the value of the *field* element's `name`[p628] attribute.
    6. If the *field* element is a `select`[p590] element, then for each `option`[p601] element in the `select`[p590] element's list of options[p593] whose selectedness[p603] is true and that is not disabled[p602], create an entry[p661] with *name* and the value[p603] of the `option`[p601] element, and append it to *entry list*.
    7. Otherwise, if the *field* element is an `input`[p539] element whose `type`[p541] attribute is in the Checkbox[p561] state or the Radio Button[p561] state:
        1. If the *field* element has a `value`[p543] attribute specified, then let *value* be the value of that attribute; otherwise, let *value* be the string "`on`".
        2. Create an entry[p661] with *name* and *value*, and append it to *entry list*.
    8. Otherwise, if the *field* element is an `input`[p539] element whose `type`[p541] attribute is in the File Upload[p563] state:
        1. If there are no selected files[p563], then create an entry[p661] with *name* and a new `File` object with an empty name, `application/octet-stream` as type, and an empty body, and append it to *entry list*.
        2. Otherwise, for each file in selected files[p563], create an entry[p661] with *name* and a `File` object representing

the file, and append it to *entry list*.

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*.

    2. Create an entry with *name* and *charset*, and append it to *entry list*.

10. Otherwise, create an entry with *name* and the value of the *field* element, and append it to *entry list*.

11. If the element has a dirname attribute, that attribute's value is not the empty string, and the element is an auto-directionality form-associated element:

    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. Create an entry with *dirname* and *dir*, and append it to *entry list*.

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*.

4.10.22.5 Selecting a form submission encoding

If the user agent is to **pick an encoding for a form**, it must run the following steps:

1. Let *encoding* be the document's character encoding.

2. If the form element has an accept-charset attribute, set *encoding* to the return value of running these substeps:

    1. Let *input* be the value of the form element's accept-charset attribute.

    2. Let *candidate encoding labels* be the result of splitting *input* on 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*.

4.10.22.6 Converting an entry list to a list of name-value pairs

The application/x-www-form-urlencoded and text/plain encoding algorithms take a list of name-value pairs, where the values must be strings, rather than an entry list where the value can be a File. The following algorithm performs the conversion.

To **convert to a list of name-value pairs** an entry list *entry list*, run these steps:

1. Let *list* be an empty list of name-value pairs.

2. For each *entry* of *entry list*:

    1. Let *name* be *entry*'s name, with 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), replaced by a string consisting of U+000D (CR) and

U+000A (LF).

2. If *entry*'s value[p661] is a `File` object, then let *value* be *entry*'s value[p661]'s `name`. Otherwise, let *value* be *entry*'s value[p661].

3. 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), in *value*, by a string consisting of U+000D (CR) and U+000A (LF).

4. Append to *list* a new name-value pair whose name is *name* and whose value is *value*.

3. Return *list*.

4.10.22.7 URL-encoded form data §[p664]

See *URL* for details on `application/x-www-form-urlencoded`. [URL][p1587]

4.10.22.8 Multipart form data §[p664]

The **`multipart/form-data` encoding algorithm**, given an entry list[p661] *entry list* and an encoding *encoding*, is as follows:

1. For each *entry* of *entry list*:

    1. 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), in *entry*'s name[p661], by a string consisting of a U+000D (CR) and U+000A (LF).

    2. If *entry*'s value[p661] is not a `File` object, 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), in *entry*'s value[p661], by a string consisting of a U+000D (CR) and U+000A (LF).

2. 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][p1586]

    - Each entry[p661] in *entry list* is a *field*, the name[p661] of the entry is the *field name* and the value[p661] 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 filenames for file fields, in the generated `multipart/form-data`[p1577] 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 filenames 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`[p1577] resource that correspond to non-file fields must not have a `Content-Type`[p102] 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`[p1577] payloads, see RFC 7578. [RFC7578][p1586]

4.10.22.9 Plain text form data §[p664]

The **`text/plain` encoding algorithm**, given a list of name-value pairs *pairs*, is as follows:

1. Let *result* be the empty string.

2. For each *pair* in *pairs*:

1. Append *pair*'s name to *result*.
2. Append a single U+003D EQUALS SIGN character (=) to *result*.
3. Append *pair*'s value to *result*.
4. 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 format is ambiguous (for example, there is no way to distinguish a literal newline in a value from the newline at the end of the value).

4.10.22.10 The SubmitEvent[p665] interface [§p665]

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)

***event*.submitter[p665]**

Returns the element representing the submit button[p532] that triggered the form submission[p657], or null if the submission was not triggered by a button.

The **submitter** attribute must return the value it was initialized to.

4.10.22.11 The FormDataEvent[p665] interface [§p665]

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)

***event*.formData[p665]**

Returns a FormData object representing names and values of elements associated to the target form[p532]. 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[p661] that is constructed[p661] when the form[p532] is submitted.

4.10.23 Resetting a form §[p666]

When a form[p532] element *form* is **reset**, run these steps:

1. Let *reset* be the result of firing an event named reset[p1576] at *form*, with the bubbles and cancelable attributes initialized to true.
2. If *reset* is true, then invoke the reset algorithm[p666] of each resettable element[p532] whose form owner[p626] is *form*.

Each resettable element[p532] 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 §[p666]

4.11.1 The details element §[p666]

Categories[p154]:

Flow content[p157].
Interactive content[p158].
Palpable content[p158].

Contexts in which this element can be used[p154]:

Where flow content[p157] is expected.

Content model[p154]:

One summary[p672] element followed by flow content[p157].

Tag omission in text/html[p154]:

Neither tag is omissible.

Content attributes[p154]:

Global attributes[p162]
name[p667] — Name of group of mutually-exclusive details[p666] elements
open[p667] — Whether the details are visible

Accessibility considerations[p154]:

For authors.
For implementers.

Sanitization[p154]:

Uncategorized[p1249].

DOM interface[p154]:

IDL
```
[Exposed=Window]
interface HTMLDetailsElement : HTMLElement {
  [HTMLConstructor] constructor();

  [CEReactions, Reflect] attribute DOMString name;
  [CEReactions, Reflect] attribute boolean open;
};
```

The details[p666] element represents[p148] a disclosure widget from which the user can obtain additional information or controls.

Note

As with all HTML elements, it is not conforming to use the details[p666] element when attempting to represent another type of control. For example, tab widgets and menu widgets are not disclosure widgets, so abusing the details[p666] element to implement these patterns is incorrect.

Note

The details[p666] element is not appropriate for footnotes. Please see the section on footnotes[p817] for details on how to mark up

footnotes.

The first summary element child of the element, if any, represents the summary or legend of the details. If there is no child summary element, the user agent should provide its own legend (e.g. "Details").

The rest of the element's contents represents the additional information or controls.

The **name** content attribute gives the name of the group of related details elements that the element is a member of. Opening one member of this group causes other members of the group to close. If the attribute is specified, its value must not be the empty string.

Before using this feature, authors should consider whether this grouping of related details elements into an exclusive accordion is helpful or harmful to users. While using an exclusive accordion can reduce the maximum amount of space that a set of content can occupy, it can also frustrate users who have to open many items to find what they want or users who want to look at the contents of multiple items at the same time.

A document must not contain more than one details element in the same details name group that has the open attribute present. Authors must not use script to add details elements to a document in a way that would cause a details name group to have more than one details element with the open attribute present.

Note

The group of elements that is created by a common name attribute is exclusive, meaning that at most one of the details elements can be open at once. While this exclusivity is enforced by user agents, the resulting enforcement immediately changes the open attributes in the markup. This requirement on authors forbids such misleading markup.

A document must not contain a details element that is a descendant of another details element in the same details name group.

Documents that use the name attribute to group multiple related details elements should keep those related elements together in a containing element (such as a section element or article element). When it makes sense for the group to be introduced with a heading, authors should put that heading in a heading element at the start of the containing element.

Note

Visually and programmatically grouping related elements together can be important for accessible user experiences. This can help users understand the relationship between such elements. When related elements are in disparate sections of a web page rather than being grouped, the elements' relationships to each other can be less discoverable or understandable.

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.

The **details name group** that contains a details element *a* also contains all the other details elements *b* that fulfill all of the following conditions:

- Both *a* and *b* are in the same tree.
- They both have a name attribute, their name attributes are not the empty string, and the value of *a*'s name attribute equals the value of *b*'s name attribute.

Every details element has a **details toggle task tracker**, which is a toggle task tracker or null, initially null.

The following attribute change steps, given *element*, *localName*, *oldValue*, *value*, and *namespace*, are used for all details elements:

1. If *namespace* is not null, then return.
2. If *localName* is name, then ensure details exclusivity by closing the given element if needed given *element*.
3. If *localName* is open:
    1. If one of *oldValue* or *value* is null and the other is not null, run the following steps, which are known as the **details notification task steps**, for this details element:

        Note

        When the open attribute is toggled several times in succession, the resulting tasks essentially get coalesced so that only one event is fired.

        1. If *oldValue* is null, queue a details toggle event task given the details element, "closed", and "open".
        2. Otherwise, queue a details toggle event task given the details element, "open", and "closed".
    2. If *oldValue* is null and *value* is not null, then ensure details exclusivity by closing other elements if needed given *element*.

The details HTML element insertion steps, given *insertedNode*, are:

1. Ensure details exclusivity by closing the given element if needed given *insertedNode*.

Note

To be clear, these attribute change and insertion steps also run when an attribute or element is inserted via the parser.

To **queue a details toggle event task** given a details element *element*, a string *oldState*, and a string *newState*:

1. If *element*'s details toggle task tracker is not null:
    1. Set *oldState* to *element*'s details toggle task tracker's old state.
    2. Remove *element*'s details toggle task tracker's task from its task queue.
    3. Set *element*'s details toggle task tracker to null.
2. Queue an element task given the DOM manipulation task source and *element* to run the following steps:
    1. Fire an event named toggle at *element*, using ToggleEvent, with the oldState attribute initialized to *oldState* and the newState attribute initialized to *newState*.
    2. Set *element*'s details toggle task tracker to null.
3. Set *element*'s details toggle task tracker to a struct with task set to the just-queued task and old state set to *oldState*.

To **ensure details exclusivity by closing other elements if needed** given a details element *element*:

1. Assert: *element* has an open attribute.
2. If *element* does not have a name attribute, or its name attribute is the empty string, then return.
3. Let *groupMembers* be a list of elements, containing all elements in *element*'s details name group except for *element*, in tree order.
4. For each element *otherElement* of *groupMembers*:
    1. If the open attribute is set on *otherElement*:
        1. Assert: *otherElement* is the only element in *groupMembers* that has the open attribute set.

2. Remove the open attribute on *otherElement*.
3. Break.

To **ensure details exclusivity by closing the given element if needed** given a details element *element*:

1. If *element* does not have an open attribute, then return.
2. If *element* does not have a name attribute, or its name attribute is the empty string, then return.
3. Let *groupMembers* be a list of elements, containing all elements in *element*'s details name group except for *element*, in tree order.
4. For each element *otherElement* of *groupMembers*:
    1. If the open attribute is set on *otherElement*:
        1. Remove the open attribute on *element*.
        2. Break.

Example

The following example shows the details element being used to hide technical details in a progress report.

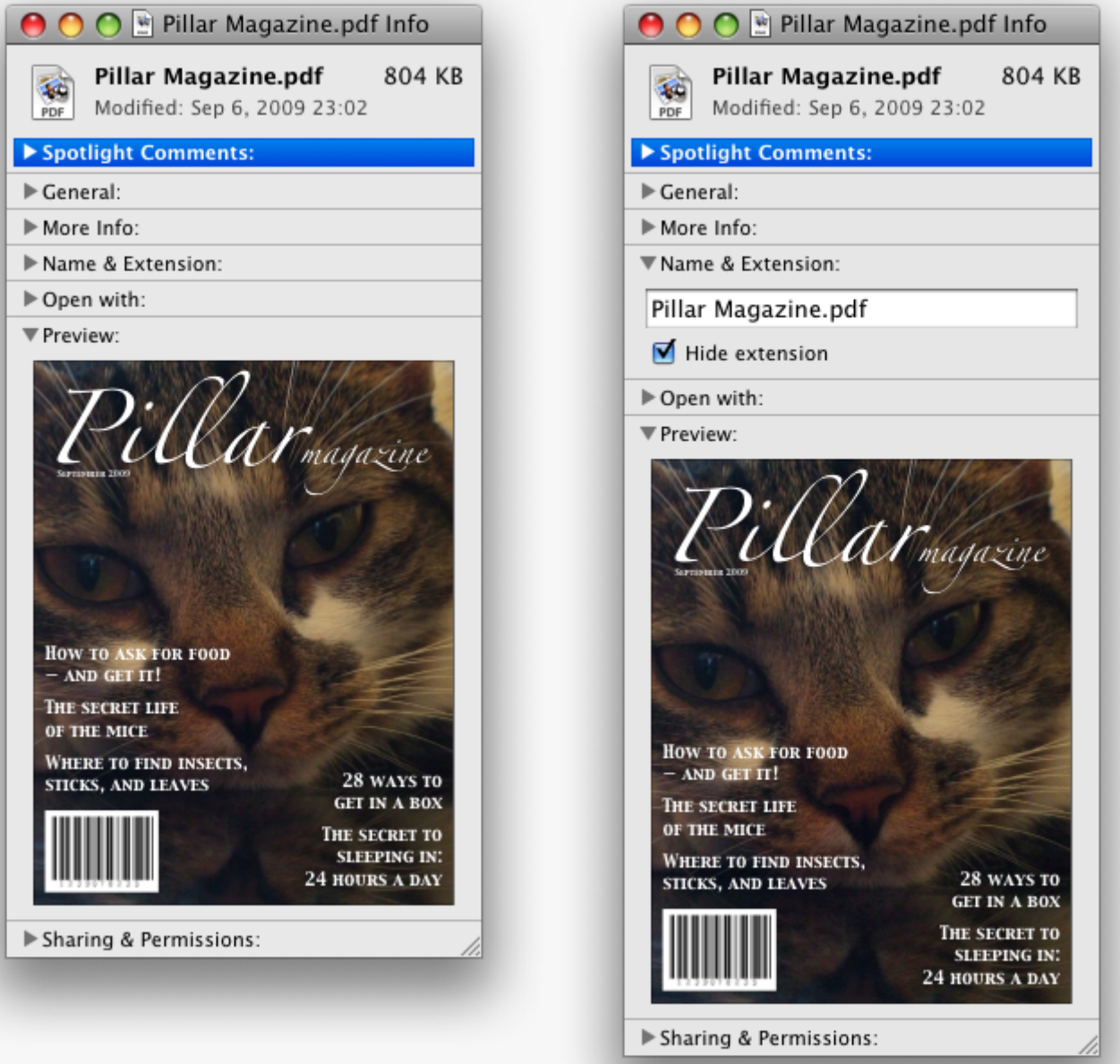
```
<section class="progress window">
 <h1>Copying "Really Achieving Your Childhood Dreams"</h1>
 <details>
  <summary>Copying... <progress max="375505392" value="97543282"></progress> 25%</summary>
  <dl>
   <dt>Transfer rate:</dt> <dd>452KB/s</dd>
   <dt>Local filename:</dt> <dd>/home/rpausch/raycd.m4v</dd>
   <dt>Remote filename:</dt> <dd>/var/www/lectures/raycd.m4v</dd>
   <dt>Duration:</dt> <dd>01:16:27</dd>
   <dt>Color profile:</dt> <dd>SD (6-1-6)</dd>
   <dt>Dimensions:</dt> <dd>320×240</dd>
  </dl>
 </details>
</section>
```

Example

The following shows how a details element can be used to hide some controls by default:

```
<details>
 <summary><label for=fn>Name & Extension:</label></summary>
 <p><input type=text id=fn name=fn value="Pillar Magazine.pdf">
 <p><label><input type=checkbox name=ext checked> Hide extension</label>
</details>
```

One could use this in conjunction with other details element 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.

¶ p67 0

Example

The following example shows the `name` p667 attribute of the `details` p666 element being used to create an exclusive accordion, a set of `details` p666 elements where a user action to open one `details` p666 element causes any open `details` p666 to close.

```
<section class="characteristics">
 <details name="frame-characteristics">
  <summary>Material</summary>
  The picture frame is made of solid oak wood.
 </details>
 <details name="frame-characteristics">
  <summary>Size</summary>
  The picture frame fits a photo 40cm tall and 30cm wide.
```

```
  The frame is 45cm tall, 35cm wide, and 2cm thick.
 </details>
 <details name="frame-characteristics">
  <summary>Color</summary>
  The picture frame is available in its natural wood
  color, or with black stain.
 </details>
</section>
```

¶ p67 1 **Example**

The following example shows what happens when the open p667 attribute is set on a details p666 element that is part of a set of elements using the name p667 attribute to create an exclusive accordion.

Given the initial markup:

```
<section class="characteristics">
 <details name="frame-characteristics" id="d1" open>...</details>
 <details name="frame-characteristics" id="d2">...</details>
 <details name="frame-characteristics" id="d3">...</details>
</section>
```

and the script:

```
document.getElementById("d2").setAttribute("open", "");
```

then the resulting tree after the script executes will be equivalent to the markup:

```
<section class="characteristics">
 <details name="frame-characteristics" id="d1">...</details>
 <details name="frame-characteristics" id="d2" open>...</details>
 <details name="frame-characteristics" id="d3">...</details>
</section>
```

because setting the open p667 attribute on `d2` removes it from `d1`.

The same happens when the user activates the summary p672 element inside of `d2`.

Example

Because the open p667 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>
 <p>Direction: North</p>
</details>
```

4.11.2 The summary element §[p67]

Categories[p154]:

None.

Contexts in which this element can be used[p154]:

As the first child of a details[p666] element.

Content model[p154]:

Phrasing content[p157], optionally intermixed with heading content[p157].

Tag omission in text/html[p154]:

Neither tag is omissible.

Content attributes[p154]:

Global attributes[p162]

Accessibility considerations[p154]:

For authors.
For implementers.

Sanitization[p154]:

Uncategorized[p1249].

DOM interface[p154]:

Uses HTMLElement[p149].

The summary[p672] element represents[p148] a summary, caption, or legend for the rest of the contents of the summary[p672] element's parent details[p666] element, if any.

A summary[p672] element is a **summary for its parent details** if the following algorithm returns true:

1. If this summary[p672] element has no parent, then return false.
2. Let *parent* be this summary[p672] element's parent.
3. If *parent* is not a details[p666] element, then return false.
4. If *parent*'s first summary[p672] element child is not this summary[p672] element, then return false.
5. Return true.

The activation behavior of summary[p672] elements is to run the following steps:

1. If this summary[p672] element is not the summary for its parent details[p672], then return.
2. Let *parent* be this summary[p672] element's parent.
3. If the open[p667] attribute is present on *parent*, then remove it. Otherwise, set *parent*'s open[p667] attribute to the empty string.

Note

This will then run the details notification task steps[p668].

4.11.3 Commands §[p67]

4.11.3.1 Facets §[p67]

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[p673].

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[p1063], or a form submission.

User agents may expose the commands[p672] that match the following criteria:

- The Hidden State[p673] facet is false (visible).
- The element is in a document with a non-null browsing context[p1046].
- Neither the element nor any of its ancestors has a `hidden`[p862] attribute specified.

User agents are encouraged to do this especially for commands that have Access Keys[p673], as a way to advertise those keys to the user.

Example

For example, such commands could be listed in the user agent's menu bar.

4.11.3.2 Using the `a` element to define a command §[p673]

An `a`[p267] element with an `href`[p314] attribute defines a command[p672].

The Label[p672] of the command is the element's descendant text content.

The Access Key[p673] of the command is the element's assigned access key[p891], if any.

The Hidden State[p673] of the command is true (hidden) if the element has a `hidden`[p862] attribute, and false otherwise.

The Disabled State[p673] facet of the command is true if the element or one of its ancestors is inert[p866], and false otherwise.

The Action[p673] of the command is to fire a `click` event[p1218] at the element.

4.11.3.3 Using the `button` element to define a command §[p673]

A `button`[p585] element always defines a command[p672].

The Label[p672], Access Key[p673], Hidden State[p673], and Action[p673] facets of the command are determined as for a elements[p673] (see the previous section).

The Disabled State[p673] of the command is true if the element or one of its ancestors is inert[p866], or if the element's disabled[p630] state is set, and false otherwise.

4.11.3.4 Using the `input` element to define a command §[p673]

An `input`[p539] element whose `type`[p541] attribute is in one of the Submit Button[p565], Reset Button[p568], Image Button[p566], Button[p569], Radio Button[p561], or Checkbox[p561] states defines a command[p672].

The Label[p672] of the command is determined as follows:

1. If the `type`[p541] attribute is in one of the Submit Button[p565], Reset Button[p568], Image Button[p566], or Button[p569] states, then the

Label[p672] is the string given by the value[p543] attribute, if any, and a UA-dependent, locale-dependent value that the UA uses to label the button itself if the attribute is absent.

2. Otherwise, if the element is a labeled control[p537], then the Label[p672] is the descendant text content of the first label[p536] element in tree order whose labeled control[p537] is the element in question. (In JavaScript terms, this is given by `element.labels[0].textContent`.)

3. Otherwise, if the value[p543] attribute is present, then the Label[p672] is the value of that attribute.

4. Otherwise, the Label[p672] is the empty string.

Note

Even though the value[p543] attribute on input[p539] elements in the Image Button[p566] state is non-conformant, the attribute can still contribute to the Label[p672] determination, if it is present and the Image Button's alt[p567] attribute is missing.

The Access Key[p673] of the command is the element's assigned access key[p891], if any.

The Hidden State[p673] of the command is true (hidden) if the element has a hidden[p862] attribute, and false otherwise.

The Disabled State[p673] of the command is true if the element or one of its ancestors is inert[p866], or if the element's disabled[p630] state is set, and false otherwise.

The Action[p673] of the command is to fire a `click` event[p1218] at the element.

4.11.3.5 Using the `option` element to define a command §[p674]

An option[p601] element with an ancestor select[p590] element and either no value[p603] attribute or a value[p603] attribute that is not the empty string defines a command[p672].

The Label[p672] of the command is the value of the option[p601] element's label[p603] attribute, if there is one, or else the option[p601] element's descendant text content, with ASCII whitespace stripped and collapsed.

The Access Key[p673] of the command is the element's assigned access key[p891], if any.

The Hidden State[p673] of the command is true (hidden) if the element has a hidden[p862] attribute, and false otherwise.

The Disabled State[p673] of the command is true if the element is disabled[p602], or if its nearest ancestor select[p590] element is disabled[p630], or if it or one of its ancestors is inert[p866], and false otherwise.

If the option[p601]'s nearest ancestor select[p590] element has a multiple[p591] attribute, the Action[p673] of the command is to toggle[p592] the option[p601] element. Otherwise, the Action[p673] is to pick[p592] the option[p601] element.

4.11.3.6 Using the `accesskey` attribute on a `legend` element to define a command §[p674]

A legend[p623] element defines a command[p672] if all of the following are true:

- It has an assigned access key[p891].
- It is a child of a fieldset[p620] element.
- Its parent has a descendant that defines a command[p672] that is neither a label[p536] element nor a legend[p623] element. This element, if it exists, is **the `legend` element's `accesskey` delegatee**.

The Label[p672] of the command is the element's descendant text content.

The Access Key[p673] of the command is the element's assigned access key[p891].

The Hidden State[p673], Disabled State[p673], and Action[p673] facets of the command are the same as the respective facets of the legend element's accesskey delegatee[p674].

Example

In this example, the legend element specifies an accesskey, which, when activated, will delegate to the input element inside the legend element.

```
<fieldset>
 <legend accesskey=p>
  <label>I want <input name=pizza type=number step=1 value=1 min=0>
   pizza(s) with these toppings</label>
 </legend>
 <label><input name=pizza-cheese type=checkbox checked> Cheese</label>
 <label><input name=pizza-ham type=checkbox checked> Ham</label>
 <label><input name=pizza-pineapple type=checkbox> Pineapple</label>
</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

closedby — Which user actions will close the dialog

open — Whether the dialog box is showing

Accessibility considerations:

For authors.

For implementers.

Sanitization[p154]**:**

Uncategorized[p1249].

DOM interface[p154]**:**

IDL

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

  [CEReactions, Reflect] attribute boolean open;
  attribute DOMString returnValue;
  [CEReactions, ReflectSetter] attribute DOMString closedBy;
  [CEReactions] undefined show();
  [CEReactions] undefined showModal();
  [CEReactions] undefined close(optional DOMString returnValue);
  [CEReactions] undefined requestClose(optional DOMString returnValue);
};
```

The dialog[p675] element represents a transitory part of an application, in the form of a small window ("dialog box"), which the user interacts with to perform a task or gather information. Once the user is done, the dialog can be automatically closed by the application, or manually closed by the user.

Especially for modal dialogs, which are a familiar pattern across all types of applications, authors should work to ensure that dialogs in their web applications behave in a way that is familiar to users of non-web applications.

Note

As with all HTML elements, it is not conforming to use the dialog[p675] element when attempting to represent another type of control. For example, context menus, tooltips, and popup listboxes are not dialog boxes, so abusing the dialog[p675] element to implement these patterns is incorrect.

An important part of user-facing dialog behavior is the placement of initial focus. The dialog focusing steps[p683] attempt to pick a good candidate for initial focus when a dialog is shown, but might not be a substitute for authors carefully thinking through the correct choice to match user expectations for a specific dialog. As such, authors should use the autofocus[p887] attribute on the descendant element of the dialog that the user is expected to immediately interact with after the dialog opens. If there is no such element, then authors should use the autofocus[p887] attribute on the dialog[p675] element itself.

Example

In the following example, a dialog is used for editing the details of a product in an inventory management web application.

```
<dialog>
  <label>Product Number <input type="text" readonly></label>
  <label>Product Name <input type="text" autofocus></label>
</dialog>
```

If the autofocus[p887] attribute was not present, the Product Number field would have been focused by the dialog focusing steps. Although that is reasonable behavior, the author determined that the more relevant field to focus was the Product Name field, as the Product Number field is readonly and expects no user input. So, the author used autofocus to override the default.

Even if the author wants to focus the Product Number field by default, they are best off explicitly specifying that by using autofocus on that input[p539] element. This makes the intent obvious to future readers of the code, and ensures the code stays robust in the face of future updates. (For example, if another developer added a close button, and positioned it in the node tree before the Product Number field).

Another important aspect of user behavior is whether dialogs are scrollable or not. In some cases, overflow (and thus scrollability) cannot be avoided, e.g., when it is caused by the user's high text zoom settings. But in general, scrollable dialogs are not expected by users. Adding large text nodes directly to dialog elements is particularly bad as this is likely to cause the dialog element itself to overflow. Authors are best off avoiding them.

Example

The following terms of service dialog respects the above suggestions.

```
<dialog style="height: 80vh;">
  <div style="overflow: auto; height: 60vh;" autofocus>
    <p>By placing an order via this Web site on the first day of the fourth month of the year
    2010 Anno Domini, you agree to grant Us a non-transferable option to claim, for now and for
    ever more, your immortal soul.</p>
    <p>Should We wish to exercise this option, you agree to surrender your immortal soul,
    and any claim you may have on it, within 5 (five) working days of receiving written
    notification from  this site or one of its duly authorized minions.</p>
    <!-- ... etc., with many more <p> elements ... -->
  </div>
  <form method="dialog">
    <button type="submit" value="agree">Agree</button>
    <button type="submit" value="disagree">Disagree</button>
  </form>
</dialog>
```

Note how the dialog focusing steps[p683] would have picked the scrollable div[p266] element by default, but similarly to the previous example, we have placed autofocus[p887] on the div[p266] so as to be more explicit and robust against future changes.

In contrast, if the p[p238] elements expressing the terms of service did not have such a wrapper div[p266] element, then the dialog[p675] itself would become scrollable, violating the above advice. Furthermore, in the absence of any autofocus[p887] attribute, such a markup pattern would have violated the above advice and tripped up the dialog focusing steps[p683]'s default behavior, and caused focus to jump to the Agree button[p585], which is a bad user experience.

Example

This dialog box has some small print. The strong[p271] element is used to draw the user's attention to the more important part.

```
<dialog>
 <h1>Add to Wallet</h1>
 <p><strong><label for=amt>How many gold coins do you want to add to your
wallet?</label></strong></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>
 <p><label><input name=round type=checkbox> Only add perfectly round coins</label></p>
 <p><input type=button onclick="submit()" value="Add Coins"></p>
</dialog>
```

The **open** attribute is a boolean attribute[p79]. When specified, it indicates that the dialog[p675] element is active and that the user can interact with it.

The **closedby** content attribute is an enumerated attribute[p80] with the following keywords and states:

Keyword	State	Brief description
any	**Any**	Close requests[p902] or clicks outside close the dialog.
closerequest	**Close Request**	Close requests[p902] close the dialog.
none	**None**	No user actions automatically close the dialog.

The closedby[p677] attribute's *invalid value default*[p80] and *missing value default*[p80] are both the **Auto** state.

Note

The Auto[p677] state behaves as Close Request[p677] state when the dialog[p675] was shown using its showModal()[p679] method; otherwise the None[p677] state.

A dialog[p675] element without an open[p677] 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[p49] implement this

requirement using the CSS rules described in the Rendering section[p1488].

¶[p678] Note

Removing the open[p677] attribute will usually hide the dialog. However, doing so has a number of strange additional consequences:

- *The close[p1575] event will not be fired.*
- *The close()[p679] method, and any close requests[p902], will no longer be able to close the dialog.*
- *If the dialog was shown using its showModal()[p679] method, the Document[p135] will still be blocked[p866].*

For these reasons, it is generally better to never remove the open[p677] attribute manually. Instead, use the requestClose()[p679] or close()[p679] methods to close the dialog, or the hidden[p862] attribute to hide it.

The tabindex[p878] attribute must not be specified on dialog[p675] elements.

For web developers (non-normative)

***dialog*.show[p678]()**

Displays the dialog[p675] element.

***dialog*.showModal[p679]()**

Displays the dialog[p675] element and makes it the top-most modal dialog.

This method honors the autofocus[p887] attribute.

***dialog*.close[p679]([*result*])**

Closes the dialog[p675] element.

The argument, if provided, provides a return value.

***dialog*.requestClose[p679]([*result*])**

Acts as if a close request[p902] was sent targeting *dialog*, by first firing a cancel[p1575] event, and if that event is not canceled with preventDefault(), proceeding to close the *dialog* in the same way as the close()[p679] method (including firing a close[p1575] event).

This is a helper utility that can be used to consolidate cancelation and closing logic into the cancel[p1575] and close[p1575] event handlers, by having all non-close request[p902] closing affordances call this method.

Note that this method ignores the closedby[p677] attribute: that is, even if closedby[p677] is set to "none[p677]", the same behavior will apply.

The argument, if provided, provides a return value.

***dialog*.returnValue[p679] [= *result*]**

Returns the dialog[p675]'s return value.

Can be set, to update the return value.

The **show()** method steps are:

1. If this has an open[p677] attribute and is modal[p680] of this is false, then return.
2. If this has an open[p677] attribute, then throw an "InvalidStateError" DOMException.
3. If the result of firing an event named beforetoggle[p1574], using ToggleEvent[p871], with the cancelable attribute initialized to true, the oldState[p872] attribute initialized to "closed", and the newState[p872] attribute initialized to "open" at this is false, then return.
4. If this has an open[p677] attribute, then return.
5. Queue a dialog toggle event task[p683] given this, "closed", "open", and null.
6. Add an open[p677] attribute to this, whose value is the empty string.
7. Set this's previously focused element[p679] to the focused[p875] element.
8. Let *document* be this's node document.

9. Let *hideUntil* be the result of running topmost popover ancestor[p933] given this, null, and false.
10. Run hide popovers until[p932] given *document*, *hideUntil*, false, and true.
11. Run the dialog focusing steps[p683] given this.

The **showModal()** method steps are to show a modal dialog[p680] given this and null.

The **close(*returnValue*)** method steps are:

1. If *returnValue* is not given, then set it to null.
2. Close the dialog[p682] this with *returnValue* and null.

The **requestClose(*returnValue*)** method steps are:

1. If *returnValue* is not given, then set it to null.
2. Request to close the dialog[p682] this with *returnValue* and null.

¶[p679] 9

Note

We use show/close as the verbs for dialog[p675] elements, as opposed to verb pairs that are more commonly thought of as antonyms such as show/hide or open/close, due to the following constraints:

- *Hiding a dialog is different from closing one. Closing a dialog gives it a return value, fires an event, unblocks the page for other dialogs, and so on. Whereas hiding a dialog is a purely visual property, and is something you can already do with the hidden[p862] attribute or by removing the open[p677] attribute. (See also the note above[p678] about removing the open[p677] attribute, and how hiding the dialog in that way is generally not desired.)*
- *Showing a dialog is different from opening one. Opening a dialog consists of creating and showing that dialog (similar to how window.open()[p969] both creates and shows a new window). Whereas showing the dialog is the process of taking a dialog[p675] element that is already in the DOM, and making it interactive and visible to the user.*
- *If we were to have a `dialog.open()` method despite the above, it would conflict with the dialog.open[p676] property.*

Furthermore, a survey of many other UI frameworks contemporary to the original design of the dialog[p675] element made it clear that the show/close verb pair was reasonably common.

In summary, it turns out that the implications of certain verbs, and how they are used in technology contexts, mean that paired actions such as showing and closing a dialog are not always expressible as antonyms.

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.

The **closedBy** getter steps are to return the keyword corresponding to the computed closed-by state[p683] given this.

Each Document[p135] has a **dialog pointerdown target**, which is an HTML dialog element[p676] or null, initially null.

Each HTML element[p46] has a **previously focused element** which is null or an element, and it is initially null. When showModal()[p679] and show()[p678] are called, this element is set to the currently focused[p875] element before running the dialog focusing steps[p683]. Elements with the popover[p925] attribute set this element to the currently focused[p875] element during the show popover algorithm[p927].

Each dialog[p675] element has a **dialog toggle task tracker**, which is a toggle task tracker[p872] or null, initially null.

Each dialog[p675] element has a **close watcher**, which is a close watcher[p904] or null, initially null.

Each dialog[p675] element has a **request close return value**, which is a string or null, initially null.

Each dialog[p675] element has a **request close source element**, which is an Element or null, initially null.

Each dialog[p675] element has an **enable close watcher for request close** boolean, initially false.

Note

dialog[p675]'s enable close watcher for request close[p679] is used to force enable the dialog's close watcher[p679] while requesting to

close[p682] it. A dialog whose computed closed-by state[p683] is the None[p677] state would otherwise have a disabled close watcher[p679].

Each `dialog`[p675] element has an **is modal** boolean, initially false.

The `dialog`[p675] HTML element insertion steps[p46], given *insertedNode*, are:

1. If *insertedNode*'s node document is not fully active[p1051], then return.
2. If *insertedNode* has an `open`[p677] attribute and is connected, then run the dialog setup steps[p683] given *insertedNode*.

The `dialog`[p675] HTML element removing steps[p46], given *removedNode*, *isSubtreeRoot*, and *oldAncestor* are:

1. If *removedNode* has an `open`[p677] attribute, then run the dialog cleanup steps[p684] given *removedNode*.
2. If *removedNode*'s node document's top layer contains *removedNode*, then remove an element from the top layer immediately given *removedNode*.
3. Set is modal[p680] of *removedNode* to false.

The following attribute change steps, given *element*, *localName*, *oldValue*, *value*, and *namespace* are used for `dialog`[p675] elements:

1. If *namespace* is not null, then return.
2. If *localName* is not `open`[p677], then return.
3. If *value* is null and *oldValue* is not null, then run the dialog cleanup steps[p684] given *element*.
4. If *element*'s node document is not fully active[p1051], then return.
5. If *element* is not connected, then return.

   Note

   This ensures that the dialog setup steps are not run on nodes that are disconnected, which would result in a close watcher[p904] being established. The dialog cleanup steps[p684] need no such guard.
6. If *value* is not null and *oldValue* is null, then run the dialog setup steps[p683] given *element*.

To **show a modal dialog** given a `dialog`[p675] element *subject* and an `Element` or null *source*:

1. If *subject* has an `open`[p677] attribute and is modal[p680] of *subject* is true, then return.
2. If *subject* has an `open`[p677] attribute, then throw an `"InvalidStateError"` `DOMException`.
3. If *subject*'s node document is not fully active[p1051], then throw an `"InvalidStateError"` `DOMException`.
4. If *subject* is not connected, then throw an `"InvalidStateError"` `DOMException`.
5. If *subject* is in the popover showing state[p926], then throw an `"InvalidStateError"` `DOMException`.
6. If the result of firing an event named `beforetoggle`[p1574], using `ToggleEvent`[p871], with the `cancelable` attribute initialized to true, the `oldState`[p872] attribute initialized to "`closed`", the `newState`[p872] attribute initialized to "`open`", and the `source`[p872] attribute initialized to *source* at *subject* is false, then return.
7. If *subject* has an `open`[p677] attribute, then return.
8. If *subject* is not connected, then return.
9. If *subject* is in the popover showing state[p926], then return.
10. Queue a dialog toggle event task[p683] given *subject*, "`closed`", "`open`", and *source*.
11. Add an `open`[p677] attribute to *subject*, whose value is the empty string.
12. Assert: *subject*'s close watcher[p679] is not null.
13. Set is modal[p680] of *subject* to true.

14. Set *subject*'s node document to be blocked by the modal dialog[p866] *subject*.

¶[p68] 1 **Note**

This will cause the focused area of the document[p875] to become inert[p866] (unless that currently focused area is a shadow-including descendant of subject*). In such cases, the focused area of the document[p875] will soon be reset[p1200] to the viewport. In a couple steps we will attempt to find a better candidate to focus.*

15. If *subject*'s node document's top layer does not already contain *subject*, then add an element to the top layer given *subject*.
16. Set *subject*'s previously focused element[p679] to the focused[p875] element.
17. Let *document* be *subject*'s node document.
18. Let *hideUntil* be the result of running topmost popover ancestor[p933] given *subject*, null, and false.
19. Run hide popovers until[p932] given *document*, *hideUntil*, false, and true.
20. Run the dialog focusing steps[p683] given *subject*.

To **set the dialog close watcher**, given a `dialog`[p675] element *dialog*:

1. Assert: *dialog*'s close watcher[p679] is null.
2. Assert: *dialog* has an `open`[p677] attribute and *dialog*'s node document is fully active[p1051].
3. Set *dialog*'s close watcher[p679] to the result of establishing a close watcher[p904] given *dialog*'s relevant global object[p1151], with:
    - *cancelAction*[p904] given *canPreventClose* being to return the result of firing an event named `cancel`[p1575] at *dialog*, with the `cancelable` attribute initialized to *canPreventClose*.
    - *closeAction*[p904] being to close the dialog[p682] given *dialog*, *dialog*'s request close return value[p679], and *dialog*'s request close source element[p679].
    - *getEnabledState*[p904] being to return true if *dialog*'s enable close watcher for request close[p679] is true or *dialog*'s computed closed-by state[p683] is not None[p677]; otherwise false.

The is valid command steps[p588] for `dialog`[p675] elements, given a `command`[p586] attribute *command*, are:

1. If *command* is in the Close[p586] state, the Request Close[p586] state, or the Show Modal[p586] state, then return true.
2. Return false.

The command steps[p588] for `dialog`[p675] elements, given an element *element*, an element *source*, and a `command`[p586] attribute *command*, are:

1. If *element* is in the popover showing[p926] state, then return.
2. If *command* is in the Close[p586] state and *element* has an `open`[p677] attribute, then close the dialog[p682] *element* with *source*'s optional value[p626] and *source*.
3. If *command* is in the Request Close[p586] state and *element* has an `open`[p677] attribute, then request to close the dialog[p682] *element* with *source*'s optional value[p626] and *source*.
4. If *command* is the Show Modal[p586] state and *element* does not have an `open`[p677] attribute, then show a modal dialog[p680] given *element* and *source*.

Example

The following buttons use `commandfor`[p586] to open and close a "confirm" `dialog`[p675] as modal when activated:

```
<button type=button
        commandfor="the-dialog"
        command="show-modal">
 Delete
</button>
<dialog id="the-dialog">
 <form action="/delete" method="POST">
```

```
  <button type="submit">
   Delete
  </button>
  <button commandfor="the-dialog"
          command="close">
   Cancel
  </button>
 </form>
</dialog>
```

When a dialog[p675] element *subject* is to be **closed**, with null or a string *result* and an Element or null *source*, run these steps:

1. If *subject* does not have an open[p677] attribute, then return.
2. Fire an event named beforetoggle[p1574], using ToggleEvent[p871], with the oldState[p872] attribute initialized to "open", the newState[p872] attribute initialized to "closed", and the source[p872] attribute initialized to *source* at *subject*.
3. If *subject* does not have an open[p677] attribute, then return.
4. Queue a dialog toggle event task[p683] given *subject*, "open", "closed", and *source*.
5. Remove *subject*'s open[p677] attribute.
6. If is modal[p680] of *subject* is true, then request an element to be removed from the top layer given *subject*.
7. Let *wasModal* be the value of *subject*'s is modal[p680] flag.
8. Set is modal[p680] of *subject* to false.
9. If *result* is not null, then set *subject*'s returnValue[p679] attribute to *result*.
10. Set *subject*'s request close return value[p679] to null.
11. Set *subject*'s request close source element[p679] to null.
12. If *subject*'s previously focused element[p679] is not null:
    1. Let *element* be *subject*'s previously focused element[p679].
    2. Set *subject*'s previously focused element[p679] to null.
    3. If *subject*'s node document's focused area of the document[p875]'s DOM anchor[p874] is a shadow-including inclusive descendant of *subject*, or *wasModal* is true, then run the focusing steps[p881] for *element*; the viewport should not be scrolled by doing this step.
13. Queue an element task[p1195] on the user interaction task source[p1204] given the *subject* element to fire an event named close[p1575] at *subject*.

To **request to close** dialog[p675] element *subject*, given null or a string *returnValue* and null or an Element *source*:

1. If *subject* does not have an open[p677] attribute, then return.
2. If *subject* is not connected or *subject*'s node document is not fully active[p1051], then return.
3. Assert: *subject*'s close watcher[p679] is not null.
4. Set *subject*'s enable close watcher for request close[p679] to true.
5. Set *subject*'s request close return value[p679] to *returnValue*.
6. Set *subject*'s request close source element[p679] to *source*.
7. Request to close[p905] *subject*'s close watcher[p679] with false.
8. Set *subject*'s enable close watcher for request close[p679] to false.

To **queue a dialog toggle event task** given a `dialog`[p675] element *element*, a string *oldState*, a string *newState*, and an `Element` or null *source*:

1. If *element*'s dialog toggle task tracker[p679] is not null:
    1. Set *oldState* to *element*'s dialog toggle task tracker[p679]'s old state[p872].
    2. Remove *element*'s dialog toggle task tracker[p679]'s task[p872] from its task queue[p1193].
    3. Set *element*'s dialog toggle task tracker[p679] to null.
2. Queue an element task[p1195] given the DOM manipulation task source[p1203] and *element* to run the following steps:
    1. Fire an event named `toggle`[p1576] at *element*, using `ToggleEvent`[p871], with the `oldState`[p872] attribute initialized to *oldState*, the `newState`[p872] attribute initialized to *newState*, and the `source`[p872] attribute initialized to *source*.
    2. Set *element*'s dialog toggle task tracker[p679] to null.
3. Set *element*'s dialog toggle task tracker[p679] to a struct with task[p872] set to the just-queued task[p1194] and old state[p872] set to *oldState*.

To retrieve a dialog's **computed closed-by state**, given a `dialog`[p675] *dialog*:

1. If the state of *dialog*'s `closedby`[p677] attribute is Auto[p677]:
    1. If *dialog*'s is modal[p680] is true, then return Close Request[p677].
    2. Return None[p677].
2. Return the state of *dialog*'s `closedby`[p677] attribute.

The **dialog focusing steps**, given a `dialog`[p675] element *subject*, are as follows:

1. If the allow focus steps[p887] given *subject*'s node document return false, then return.
2. Let *control* be null.
3. If *subject* has the `autofocus`[p887] attribute, then set *control* to *subject*.
4. If *control* is null, then set *control* to the focus delegate[p880] of *subject*.
5. If *control* is null, then set *control* to *subject*.
6. Run the focusing steps[p881] for *control*.

    Note

    If control *is not focusable[p876], this will do nothing. This would only happen if subject had no focus delegate, and the user agent decided that* `dialog`[p675] *elements were not generally focusable. In that case, any earlier modifications[p681] to the focused area of the document[p875] will apply.*

7. Let *topDocument* be *control*'s node navigable[p1036]'s top-level traversable[p1037]'s active document[p1036].
8. If *control*'s node document's origin is not the same[p940] as the origin of *topDocument*, then return.
9. Empty *topDocument*'s autofocus candidates[p888].
10. Set *topDocument*'s autofocus processed flag[p888] to true.

The **dialog setup steps**, given a `dialog`[p675] element *subject*, are as follows:

1. Assert: *subject* has an `open`[p677] attribute.
2. Assert: *subject* is connected.
3. Assert: *subject*'s node document's open dialogs list[p137] does not contain *subject*.
4. Add *subject* to *subject*'s node document's open dialogs list[p137].
5. Set the dialog close watcher[p681] with *subject*.

The **dialog cleanup steps**, given a dialog element *subject*, are as follows:

1. Remove *subject* from *subject*'s node document's open dialogs list.
2. If *subject*'s close watcher is not null:
    1. Destroy *subject*'s close watcher.
    2. Set *subject*'s close watcher to null.

4.11.5 Dialog light dismiss

Note

"Light dismiss" means that clicking outside of a dialog element whose closedby attribute is in the Any state will close the dialog element. This is in addition to how such dialogs respond to close requests.

To **light dismiss open dialogs**, given a PointerEvent *event*:

1. Assert: *event*'s isTrusted attribute is true.
2. Let *document* be *event*'s target's node document.
3. If *document*'s open dialogs list is empty, then return.
4. Let *ancestor* be the result of running nearest clicked dialog given *event*.
5. If *event*'s type is "pointerdown", then set *document*'s dialog pointerdown target to *ancestor*.
6. If *event*'s type is "pointerup":
    1. Let *sameTarget* be true if *ancestor* is *document*'s dialog pointerdown target.
    2. Set *document*'s dialog pointerdown target to null.
    3. If *sameTarget* is false, then return.
    4. Let *topmostDialog* be the last element of *document*'s open dialogs list.
    5. If *ancestor* is *topmostDialog*, then return.
    6. If *topmostDialog*'s computed closed-by state is not Any, then return.
    7. Assert: *topmostDialog*'s close watcher is not null.
    8. Request to close *topmostDialog*'s close watcher with false.

To **run light dismiss activities**, given a PointerEvent *event*:

1. Run light dismiss open popovers with *event*.
2. Run light dismiss open dialogs with *event*.

Run light dismiss activities will be called by the Pointer Events spec when the user clicks or touches anywhere on the page.

To find the **nearest clicked dialog**, given a PointerEvent *event*:

1. Let *target* be *event*'s target.
2. If *target* is a dialog element, *target* has an open attribute, *target*'s is modal is true, and *event*'s clientX and clientY are outside the bounds of *target*, then return null.

Note

The check for clientX and clientY is because a pointer event that hits the ::backdrop *pseudo element of a dialog will result in* event *having a target of the dialog element itself.*

3. Let *currentNode* be *target*.
4. While *currentNode* is not null:
    1. If *currentNode* is a `dialog` element and *currentNode* has an `open` attribute, then return *currentNode*.
    2. Set *currentNode* to *currentNode*'s parent in the flat tree.
5. Return null.

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 a 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.
Script-supporting element.

Contexts in which this element can be used:

Where metadata content is expected.
Where phrasing content is expected.
Where script-supporting elements are expected.

Content model:

If there is no `src` attribute, depends on the value of the `type` attribute, but must match script content restrictions.
If there *is* a `src` attribute, the element must be either empty or contain only script documentation that also matches script content restrictions.

Tag omission in text/html:

Neither tag is omissible.

Content attributes:

Global attributes
`type` — Type of script
`src` — Address of the resource
`nomodule` — Prevents execution in user agents that support module scripts
`async` — Execute script when available, without blocking while fetching
`defer` — Defer script execution
`blocking` — Whether the element is potentially render-blocking
`crossorigin` — How the element handles crossorigin requests
`referrerpolicy` — Referrer policy for fetches initiated by the element
`integrity` — Integrity metadata used in *Subresource Integrity* checks [SRI]

fetchpriority[p688] — Sets the priority for fetches initiated by the element

Accessibility considerations[p154]:

For authors.
For implementers.

Sanitization[p154]:

Unsafe[p1249].

DOM interface[p154]:

IDL

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

  [CEReactions, Reflect] attribute DOMString type;
  [CEReactions, ReflectURL] attribute USVString src;
  [CEReactions, Reflect] attribute boolean noModule;
  [CEReactions] attribute boolean async;
  [CEReactions, Reflect] attribute boolean defer;
  [SameObject, PutForwards=value, Reflect] readonly attribute DOMTokenList blocking;
  [CEReactions] attribute DOMString? crossOrigin;
  [CEReactions] attribute DOMString referrerPolicy;
  [CEReactions, Reflect] attribute DOMString integrity;
  [CEReactions] attribute DOMString fetchPriority;

  [CEReactions] attribute DOMString text;

  static boolean supports(DOMString type);

  // also has obsolete members
};
```

The script[p685] element allows authors to include dynamic script, instructions to the user agent, and data blocks in their documents. The element does not represent[p148] content for the user.

The script element has two core attributes. 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[p1153], to be interpreted according to the JavaScript *Script* top-level production. Authors should omit the type[p686] attribute instead of redundantly setting it.
- Setting the attribute to an ASCII case-insensitive match for "`module`" means that the script is a JavaScript module script[p1153], to be interpreted according to the JavaScript *Module* top-level production.
- Setting the attribute to an ASCII case-insensitive match for "`importmap`" means that the script is an import map[p1177], containing JSON that will be used to control the behavior of module specifier resolution[p1171].
- Setting the attribute to an ASCII case-insensitive match for "`speculationrules`" means that the script defines a speculation rule set[p1120], containing JSON that will be used to describe speculative loads[p1118].
- Setting the attribute to any other value means that the script is a **data block**, which is not processed by the user agent, but instead by author script or other tools. 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[p686] must be denoted using a valid MIME type string is in place to avoid potential future collisions. Values for the type[p686] attribute that are not MIME types, like "`module`" or "`importmap`", are used by the standard to denote types of scripts which have special behavior in user agents. 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.

The second core attribute is the **src** attribute. It must only be specified for classic scripts[p1153] and JavaScript module scripts[p1153], and denotes that instead of using the element's child text content as the script content, the script will be fetched from the specified URL. If src[p686] is specified, it must be a valid non-empty URL potentially surrounded by spaces[p100].

Which other attributes may be specified on a given `script`[p685] element is determined by the following table:

	`nomodule`[p687]	`async`[p687]	`defer`[p687]	`blocking`[p688]	`crossorigin`[p688]	`referrerpolicy`[p688]	`integrity`[p688]	`fetchpriority`[p688]
External classic scripts	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes
Inline classic scripts	Yes	·	·	·	Yes*	Yes*	·	·†
External module scripts	·	Yes	·	Yes	Yes	Yes	Yes	Yes
Inline module scripts	·	Yes	·	·	Yes*	Yes*	·	·†
Import maps	·	·	·	·	·	·	·	·
Speculation rules	·	·	·	·	·	·	·	·
Data blocks	·	·	·	·	·	·	·	·

* Although inline scripts have no initial fetches, the `crossorigin`[p688] and `referrerpolicy`[p688] attribute on inline scripts affects the credentials mode and referrer policy used by module imports, including dynamic `import()`.

† Unlike `crossorigin`[p688] and `referrerpolicy`[p688], `fetchpriority`[p688] does not affect module imports. See some discussion in issue #10276.

The contents of inline `script`[p685] elements, or the external script resource, must conform with the requirements of the JavaScript specification's *Script* or *Module* productions, for classic scripts[p1153] and JavaScript module scripts[p1153] respectively. [JAVASCRIPT][p1583]

The contents of inline `script`[p685] elements for import maps[p1177] must conform with the import map authoring requirements[p1176].

The contents of inline `script`[p685] elements for speculation rule sets[p1120] must conform with the speculation rule set authoring requirements[p1119].

When used to include data blocks[p686], the data must be embedded inline, the format of the data must be given using the `type`[p686] attribute, and the contents of the `script`[p685] element must conform to the requirements defined for the format used.

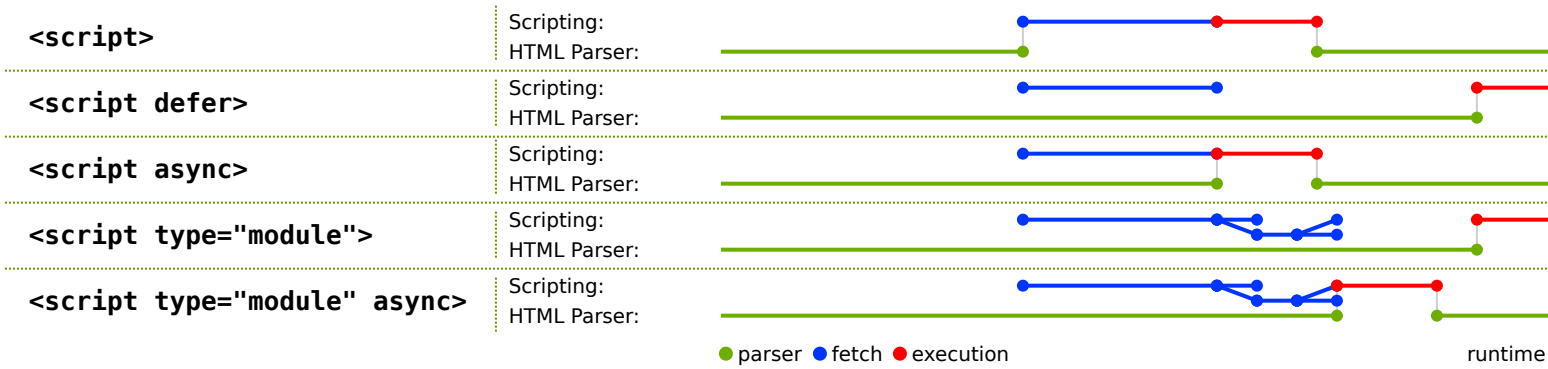
The **`nomodule`** attribute is a boolean attribute[p79] that prevents a script from being executed in user agents that support module scripts[p1153]. This allows selective execution of module scripts[p1153] in modern user agents and classic scripts[p1153] in older user agents, as shown below[p690].

The **`async`** and **`defer`** attributes are boolean attributes[p79] that indicate how the script should be evaluated. There are several possible modes that can be selected using these attributes, depending on the script's type.

For external classic scripts[p1153], if the `async`[p687] attribute is present, then the classic script will be fetched in parallel[p44] to parsing and evaluated as soon as it is available (potentially before parsing completes). If the `async`[p687] attribute is not present but the `defer`[p687] attribute is present, then the classic script will be fetched in parallel[p44] 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[p1153], if the `async`[p687] attribute is present, then the module script and all its dependencies will be fetched in parallel[p44] 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[p44] to parsing and evaluated when the page has finished parsing. (The `defer`[p687] attribute has no effect on module scripts.)

This is all summarized in the following schematic diagram:



Note

The exact processing details for these attributes are, for mostly historical reasons, somewhat non-trivial, involving a number of aspects of HTML. The implementation requirements are therefore by necessity scattered throughout the specification. The algorithms below[p692] describe the core of this processing, but these algorithms reference and are referenced by the parsing rules for `script`[p685] start[p1428] and end[p1444] tags in HTML, in foreign content[p1457], and in XML[p1485], the rules for the `document.write()`[p1223] method, the handling of scripting[p1140], etc.

Note

When inserted using the `document.write()`[p1223] method, `script`[p685] elements usually[p1428] execute (typically blocking further script execution or HTML parsing). When inserted using the `innerHTML`[p1229] and `outerHTML`[p1230] attributes, they do not execute at all.

The `defer`[p687] attribute may be specified even if the `async`[p687] attribute is specified, to cause legacy web browsers that only support `defer`[p687] (and not `async`[p687]) to fall back to the `defer`[p687] behavior instead of the blocking behavior that is the default.

The **`blocking`** attribute is a blocking attribute[p107].

The **`crossorigin`** attribute is a CORS settings attribute[p103]. For external classic scripts[p1153], it controls whether error information will be exposed, when the script is obtained from other origins[p939]. For external module scripts[p1153], it controls the credentials mode used for the initial fetch of the module source, if cross-origin. For both classic[p1153] and module scripts[p1153], it controls the credentials mode used for cross-origin module imports.

Note

Unlike classic scripts[p1153], module scripts[p1153] require the use of the CORS protocol for cross-origin fetching.

The **`referrerpolicy`** attribute is a referrer policy attribute[p104]. Its sets the referrer policy used for the initial fetch of an external script, as well as the fetching of any imported module scripts. [REFERRERPOLICY][p1585]

Example

An example of a `script`[p685] element's referrer policy being used when fetching imported scripts but not other subresources:

```
<script referrerpolicy="origin">
  fetch('/api/data');    // not fetched with <script>'s referrer policy
  import('./utils.mjs'); // is fetched with <script>'s referrer policy ("origin" in this case)
</script>
```

The **`integrity`** attribute sets the integrity metadata used for the initial fetch of an external script. The value must match the requirements of the integrity attribute. [SRI][p1586]

The **`fetchpriority`** attribute is a fetch priority attribute[p108]. Its sets the priority used for the initial fetch of an external script.

Changing any of these attributes dynamically has no direct effect; these attributes are only used at specific times described in the processing model[p692].

The **`crossOrigin`** IDL attribute must reflect[p108] the `crossorigin`[p688] content attribute, limited to only known values[p109].

The **`referrerPolicy`** IDL attribute must reflect[p108] the `referrerpolicy`[p688] content attribute, limited to only known values[p109].

The **`fetchPriority`** IDL attribute must reflect[p108] the `fetchpriority`[p688] content attribute, limited to only known values[p109].

The **`async`** getter steps are:

1. If this's force async[p692] is true, then return true.
2. If this's `async`[p687] content attribute is present, then return true.
3. Return false.

The `async`[p688] setter steps are:

1. Set this's force async[p692] to false.

2. If the given value is true, then set this's async[p687] content attribute to the empty string.
3. Otherwise, remove this's async[p687] content attribute.

For web developers (non-normative)

***script*.text[p689] [= *value*]**

Returns the child text content of the element.

script*.text[p689] = *value

Replaces the element's children with the text given by *value*.

HTMLScriptElement[p686].supports[p689](*type*)

Returns true if the given *type* is a script type supported by the user agent. The possible script types in this specification are "classic", "module", and "importmap", but others might be added in the future.

The **text** getter steps are to return this's child text content.

The text[p689] setter steps are to string replace all with the given value within this.

The static **supports(*type*)** method steps are:

✔ MDN

1. If *type* is "classic", then return true.
2. If *type* is "module", then return true.
3. If *type* is "importmap", then return true.
4. If *type* is "speculationrules", then return true.
5. Return false.

Note

The type argument has to exactly match these values; we do not perform an ASCII case-insensitive match. This is different from how type[p686] content attribute values are treated, and how DOMTokenList's supports() method works, but it aligns with the WorkerType[p1332] enumeration used in the Worker()[p1333] constructor.

Example

In this example, two script[p685] elements are used. One embeds an external classic script[p1153], and the other includes some data as a data block[p686].

```
<script src="game-engine.js"></script>
<script type="text/x-game-map">
........U.........e
o............A....e
.....A.....AAA....e
.A..AAA...AAAAA...e
</script>
```

The data in this case might be used by the script to generate the map of a video game. The data doesn't have to be used that way, though; maybe the map data is actually embedded in other parts of the page's markup, and the data block here is just used by the site's search engine to help users who are looking for particular features in their game maps.

Example

The following sample shows how a script[p685] element can be used to define a function that is then used by other parts of the document, as part of a classic script[p1153]. It also shows how a script[p685] element can be used to invoke script while the document is being parsed, in this case to initialize the form's output.

```
<script>
 function calculate(form) {
   var price = 52000;
```

```
    if (form.elements.brakes.checked)
      price += 1000;
    if (form.elements.radio.checked)
      price += 2500;
    if (form.elements.turbo.checked)
      price += 5000;
    if (form.elements.sticker.checked)
      price += 250;
    form.elements.result.value = price;
  }
</script>
<form name="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>
```

¶ p69 0

Example

The following sample shows how a `script` p685 element can be used to include an external JavaScript module script p1153.

```
<script type="module" src="app.mjs"></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` p685 element in the same `Window` p965 imports the module from `app.mjs` (e.g. via `import "./app.mjs";`), then the same JavaScript module script p1153 created by the former `script` p685 element will be imported.

¶ p69 0

Example

This example shows how to include a JavaScript module script p1153 for modern user agents, and a classic script p1153 for older user agents:

```
<script type="module" src="app.mjs"></script>
<script nomodule defer src="classic-app-bundle.js"></script>
```

In modern user agents that support JavaScript module scripts p1153, the `script` p685 element with the `nomodule` p687 attribute will be ignored, and the `script` p685 element with a `type` p686 of "`module`" will be fetched and evaluated (as a JavaScript module script p1153). Conversely, older user agents will ignore the `script` p685 element with a `type` p686 of "`module`", as that is an unknown script type for them — but they will have no problem fetching and evaluating the other `script` p685 element (as a classic script p1153), since they do not implement the `nomodule` p687 attribute.

¶ p69 0

Example

The following sample shows how a `script` p685 element can be used to write an inline JavaScript module script p1153 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]p1588

```
<script type="module">
 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);
</script>
```

Some notable features gained by using a JavaScript 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 objectp1145. Also note that no matter where this scriptp685 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.

¶p69 **Example**

1 The following sample shows how a JSON module scriptp1153 can be imported from inside a JavaScript module scriptp1153:

```
<script type="module">
 import peopleInSpace from "http://api.open-notify.org/astros.json" with { type: "json" };

 const list = document.querySelector("#people-in-space");
 for (const { craft, name } of peopleInSpace.people) {
   const li = document.createElement("li");
   li.textContent = `${name} / ${craft}`;
   list.append(li);
 }
</script>
```

MIME type checking for module scripts is strict. In order for the fetch of the JSON module scriptp1153 to succeed, the HTTP response must have a JSON MIME type, for example `Content-Type: text/json`. On the other hand, if the `with { type: "json" }` part of the statement is omitted, it is assumed that the intent is to import a JavaScript module scriptp1153, and the fetch will fail if the HTTP response has a MIME type that is not a JavaScript MIME type.

4.12.1.1 Processing model §[p692]

A script[p685] element has several associated pieces of state.

A script[p685] element has a **parser document**, which is either null or a Document[p135], initially null. It is set by the HTML parser[p1368] and the XML parser[p1484] on script[p685] elements they insert, and affects the processing of those elements. script[p685] elements with non-null parser documents[p692] are known as **parser-inserted**.

A script[p685] element has a **preparation-time document**, which is either null or a Document[p135], initially null. It is used to prevent scripts that move between documents during preparation[p694] from executing[p698].

A script[p685] element has a **force async** boolean, initially true. It is set to false by the HTML parser[p1368] and the XML parser[p1484] on script[p685] elements they insert, and when the element gets an async[p687] content attribute added.

A script[p685] element has a **from an external file** boolean, initially false. It is determined when the script is prepared[p694], based on the src[p686] attribute of the element at that time.

A script[p685] element has a **ready to be parser-executed** boolean, initially false. This is used only used for elements that are also parser-inserted[p692], to let the parser know when to execute the script.

A script[p685] element has an **already started** boolean, initially false.

A script[p685] element has a **delaying the load event** boolean, initially false.

A script[p685] element has a **type**, which is either null, "`classic`", "`module`", "`importmap`", or "`speculationrules`", initially null. It is determined when the element is prepared[p694], based on the type[p686] attribute of the element at that time.

A script[p685] element has a **result**, which is either "`uninitialized`", null (representing an error), a script[p1152], an import map parse result[p1170], or a speculation rules parse result[p1170]. It is initially "`uninitialized`".

A script[p685] element has **steps to run when the result is ready**, which are a series of steps or null, initially null. To **mark as ready** a script[p685] element *el* given a *result*:

1. Set *el*'s result[p692] to *result*.
2. If *el*'s steps to run when the result is ready[p692] are not null, then run them.
3. Set *el*'s steps to run when the result is ready[p692] to null.
4. Set *el*'s delaying the load event[p692] to false.

A script[p685] element *el* is implicitly potentially render-blocking[p107] if *el*'s type[p692] is "`classic`", *el* is parser-inserted[p692], and *el* does not have an async[p687] or defer[p687] attribute.

The cloning steps for script[p685] elements given *node*, *copy*, and *subtree* are to set *copy*'s already started[p692] to *node*'s already started[p692].

When an async[p687] attribute is added to a script[p685] element *el*, the user agent must set *el*'s force async[p692] to false.

Whenever a script[p685] element *el*'s delaying the load event[p692] is true, the user agent must delay the load event[p1458] of *el*'s preparation-time document[p692].

The script[p685] HTML element post-connection steps[p46], given *insertedNode*, are:

1. If *insertedNode* is parser-inserted[p692], then return.
2. Prepare the script element[p694] given *insertedNode*.

Example

The HTML element post-connection steps[p46] only run when the inserted element is still connected, which protects against cases where an earlier-inserted script[p685] removes a later-inserted script[p685]. For instance:

```
<script>
const script1 = document.createElement('script');
script1.innerText = `
  document.querySelector('#script2').remove();
`;

const script2 = document.createElement('script');
script2.id = 'script2';
script2.textContent = `console.log('script#2 running')`;

document.body.append(script1, script2);
</script>
```

Nothing is printed to the console in this example. By the time the HTML element post-connection steps[p46] run for the first script[p685] that was atomically inserted by append(), it can observe that the second script[p685] is already connected to the DOM, and it removes it from the DOM. Because the second script[p685] is no longer connected, its HTML element post-connection steps[p46] do not run, and it does not get prepared[p694].

The script[p685] HTML element removing steps[p46] given *removedNode* are:

1. If *removedNode*'s result[p692] is a speculation rules parse result[p1170]:
    1. Unregister speculation rules[p1171] given *removedNode*'s relevant global object[p1151] and *removedNode*'s result[p692].
    2. Set *removedNode*'s already started[p692] to false.
    3. Set *removedNode*'s result[p692] to null.

The script[p685] children changed steps given *changedNode* are:

1. If the script[p685] element is not connected, then return.
2. Run the script[p685] HTML element post-connection steps[p46], given *changedNode*.

Example

This has an interesting implication on the execution order of a script[p685] element and any newly-inserted child script[p685] elements. Consider the following snippet:

```
<script id=outer-script></script>

<script>
  const outerScript = document.querySelector('#outer-script');

  const start = new Text('console.log(1);');
  const innerScript = document.createElement('script');
  innerScript.textContent = `console.log('inner script executing')`;
  const end = new Text('console.log(2);');

  outerScript.append(start, innerScript, end);

  // Logs:
  // 1
  // 2
  // inner script executing
</script>
```

By the time the second script block executes, the `outer-script` has already been prepared[p694], but because it is empty, it did not execute and therefore is not marked as already started[p692]. The atomic insertion of the Text nodes and nested script[p685] element have the following effects:

1. All three child nodes get atomically inserted as children of `outer-script`; all of their insertion steps run, which have no observable consequences in this case.
2. The `outer-script`'s children changed steps run, which prepares[p694] that script; because its body is now non-empty, this executes the contents of the two `Text` nodes, in order.
3. The `script`[p685] HTML element post-connection steps[p46] finally run for `innerScript`, causing its body to execute.

The following attribute change steps, given *element*, *localName*, *oldValue*, *value*, and *namespace*, are used for all `script`[p685] elements:

1. If *namespace* is not null, then return.
2. If *localName* is `src`[p686], *value* is not null, and *element* is connected, then run the `script`[p685] HTML element post-connection steps[p46], given *element*.

To **prepare the script element** given a `script`[p685] element *el*:

1. If *el*'s already started[p692] is true, then return.
2. Let *parser document* be *el*'s parser document[p692].
3. Set *el*'s parser document[p692] to null.

   Note

   This is done so that if parser-inserted `script`[p685] 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 *el* does not have an `async`[p687] attribute, then set *el*'s force async[p692] to true.

   Note

   This is done so that if a parser-inserted `script`[p685] 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 an async fashion even if the `async`[p687] attribute isn't set.
5. Let *source text* be *el*'s child text content.
6. If *el* has no `src`[p686] attribute, and *source text* is the empty string, then return.
7. If *el* is not connected, then return.
8. If any of the following are true:
   - *el* has a `type`[p686] attribute whose value is the empty string;
   - *el* has no `type`[p686] attribute but it has a `language`[p1533] attribute and *that* attribute's value is the empty string; or
   - *el* has neither a `type`[p686] attribute nor a `language`[p1533] attribute,

   then let *the script block's type string* for this `script`[p685] element be "`text/javascript`".

   Otherwise, if *el* has a `type`[p686] attribute, then let *the script block's type string* be the value of that attribute.

   Otherwise, *el* has a non-empty `language`[p1533] attribute; let *the script block's type string* be the concatenation of "`text/`" and the value of *el*'s `language`[p1533] attribute.

   Note

   The `language`[p1533] attribute is never conforming, and is always ignored if there is a `type`[p686] attribute present.
9. If *the script block's type string* is a JavaScript MIME type essence match, or *el* has a `type`[p686] attribute whose value with leading and trailing ASCII whitespace stripped is a JavaScript MIME type essence match, then set *el*'s type[p692] to "`classic`".
10. Otherwise, if *the script block's type string* is an ASCII case-insensitive match for the string "`module`", then set *el*'s type[p692] to "`module`".

11. Otherwise, if *the script block's type string* is an ASCII case-insensitive match for the string "`importmap`", then set *el*'s type to "`importmap`".

12. Otherwise, if *the script block's type string* is an ASCII case-insensitive match for the string "`speculationrules`", then set *el*'s type to "`speculationrules`".

13. Otherwise, return. (No script is executed, and *el*'s type is left as null.)

14. If *parser document* is non-null, then set *el*'s parser document back to *parser document* and set *el*'s force async to false.

15. Set *el*'s already started to true.

16. Set *el*'s preparation-time document to its node document.

17. If *parser document* is non-null, and *parser document* is not equal to *el*'s preparation-time document, then return.

18. If scripting is disabled for *el*, then return.

    > **Note**
    >
    > *The definition of scripting is disabled means that, amongst others, the following scripts will not execute: scripts in `XMLHttpRequest`'s `responseXML` documents, scripts in `DOMParser`-created documents, scripts in documents created by `XSLTProcessor`'s `transformToDocument` feature, and scripts that are first inserted by a script into a `Document` that was created using the `createDocument()` API. [XHR] [DOMPARSING] [XSLTP] [DOM]*

19. If *el* has a `nomodule` content attribute and its type is "`classic`", then return.

    > **Note**
    >
    > *This means specifying `nomodule` on a module script has no effect; the algorithm continues onward.*

20. Let *cspType* be "`script speculationrules`" if *el*'s type is "`speculationrules`"; otherwise, "`script`".

21. If *el* does not have a `src` content attribute, and the Should element's inline behavior be blocked by Content Security Policy? algorithm returns "`Blocked`" when given *el*, *cspType*, and *source text*, then return. [CSP]

22. If *el* has an `event` attribute and a `for` attribute, and *el*'s type is "`classic`":

    1. Let *for* be the value of *el*'s `for` attribute.
    2. Let *event* be the value of *el*'s `event` attribute.
    3. Strip leading and trailing ASCII whitespace from *event* and *for*.
    4. If *for* is not an ASCII case-insensitive match for the string "`window`", then return.
    5. If *event* is not an ASCII case-insensitive match for either the string "`onload`" or the string "`onload()`", then return.

23. If *el* has a `charset` attribute, then let *encoding* be the result of getting an encoding from the value of the `charset` attribute.

    If *el* does not have a `charset` attribute, or if getting an encoding failed, then let *encoding* be *el*'s node document's the encoding.

    > **Note**
    >
    > *If el's type is "`module`", this encoding will be ignored.*

24. Let *classic script CORS setting* be the current state of *el*'s `crossorigin` content attribute.

25. Let *module script credentials mode* be the CORS settings attribute credentials mode for *el*'s `crossorigin` content attribute.

26. Let *cryptographic nonce* be *el*'s [[CryptographicNonce]] internal slot's value.

27. If *el* has an `integrity` attribute, then let *integrity metadata* be that attribute's value.

    Otherwise, let *integrity metadata* be the empty string.

28. Let *referrer policy* be the current state of *el*'s `referrerpolicy` content attribute.

29. Let *fetch priority* be the current state of *el*'s `fetchpriority`[p688] content attribute.

30. Let *parser metadata* be "`parser-inserted`" if *el* is parser-inserted[p692], and "`not-parser-inserted`" otherwise.

31. Let *options* be a script fetch options[p1154] whose cryptographic nonce[p1154] is *cryptographic nonce*, integrity metadata[p1154] is *integrity metadata*, parser metadata[p1154] is *parser metadata*, credentials mode[p1154] is *module script credentials mode*, referrer policy[p1154] is *referrer policy*, and fetch priority[p1154] is *fetch priority*.

32. Let *settings object* be *el*'s node document's relevant settings object[p1151].

33. If *el* has a `src`[p686] content attribute:

    1. If *el*'s type[p692] is "`importmap`" or "`speculationrules`", then queue an element task[p1195] on the DOM manipulation task source[p1203] given *el* to fire an event named `error`[p1575] at *el*, and return.

       Note

       External import maps and speculation rules are not currently supported. See WICG/import-maps issue #235 and WICG/nav-speculation issue #348 for discussions on adding support.

    2. Let *src* be the value of *el*'s `src`[p686] attribute.

    3. If *src* is the empty string, then queue an element task[p1195] on the DOM manipulation task source[p1203] given *el* to fire an event named `error`[p1575] at *el*, and return.

    4. Set *el*'s from an external file[p692] to true.

    5. Let *url* be the result of encoding-parsing a URL[p101] given *src*, relative to *el*'s node document.

    6. If *url* is failure, then queue an element task[p1195] on the DOM manipulation task source[p1203] given *el* to fire an event named `error`[p1575] at *el*, and return.

    7. If *el* is potentially render-blocking[p107], then block rendering[p142] on *el*.

    8. Set *el*'s delaying the load event[p692] to true.

    9. If *el* is currently render-blocking[p142], then set *options*'s render-blocking[p1154] to true.

    10. Let *onComplete* given *result* be the following steps:

        1. Mark as ready[p692] *el* given *result*.

    11. Switch on *el*'s type[p692]:

        ↪ "`classic`"

        Fetch a classic script[p1155] given *url*, *settings object*, *options*, *classic script CORS setting*, *encoding*, and *onComplete*.

        ↪ "`module`"

        If *el* does not have an `integrity`[p688] attribute, then set *options*'s integrity metadata[p1154] to the result of resolving a module integrity metadata[p1155] with *url* and *settings object*.

        Fetch an external module script graph[p1157] given *url*, *settings object*, *options*, and *onComplete*.

    For performance reasons, user agents may start fetching the classic script or module graph (as defined above) as soon as the `src`[p686] attribute is set, instead, in the hope that *el* will become connected (and that the `crossorigin`[p688] attribute won't change value in the meantime). Either way, once *el* becomes connected[p47], the load must have started as described in this step. If the UA performs such prefetching, but *el* never becomes connected, or the `src`[p686] attribute is dynamically changed, or the `crossorigin`[p688] attribute is dynamically changed, then the user agent will not execute the script so obtained, and the fetching process will have been effectively wasted.

34. If *el* does not have a `src`[p686] content attribute:

    1. Let *base URL* be *el*'s node document's document base URL[p101].

    2. Switch on *el*'s type[p692]:

↪ "`classic`"

1. Let *script* be the result of creating a classic script[p1161] using *source text*, *settings object*, *base URL*, and *options*.
2. Mark as ready[p692] *el* given *script*.

↪ "`module`"

1. Set *el*'s delaying the load event[p692] to true.
2. If *el* is potentially render-blocking[p107]:
    1. Block rendering[p142] on *el*.
    2. Set *options*'s render-blocking[p1154] to true.
3. Fetch an inline module script graph[p1158], given *source text*, *base URL*, *settings object*, *options*, and with the following steps given *result*:
    1. Queue an element task[p1195] on the networking task source[p1204] given *el* to perform the following steps:
        1. Mark as ready[p692] *el* given *result*.

    Note

    Queueing a task here means that, even if the inline module script has no dependencies or synchronously results in a parse error, we won't proceed to execute the script element[p698] synchronously.

↪ "`importmap`"

1. Let *result* be the result of creating an import map parse result[p1170] given *source text* and *base URL*.
2. Mark as ready[p692] *el* given *result*.

↪ "`speculationrules`"

1. Let *result* be the result of creating a speculation rules parse result[p1170] given *source text* and *el*'s node document.
2. Mark as ready[p692] *el* given *result*.

35. If *el*'s type[p692] is "`classic`" and *el* has a `src`[p686] attribute, or *el*'s type[p692] is "`module`":
    1. Assert: *el*'s result[p692] is "`uninitialized`".
    2. If *el* has an `async`[p687] attribute or *el*'s force async[p692] is true:
        1. Let *scripts* be *el*'s preparation-time document[p692]'s set of scripts that will execute as soon as possible[p698].
        2. Append *el* to *scripts*.
        3. Set *el*'s steps to run when the result is ready[p692] to the following:
            1. Execute the script element[p698] *el*.
            2. Remove *el* from *scripts*.
    3. Otherwise, if *el* is not parser-inserted[p692]:
        1. Let *scripts* be *el*'s preparation-time document[p692]'s list of scripts that will execute in order as soon as possible[p698].
        2. Append *el* to *scripts*.
        3. Set *el*'s steps to run when the result is ready[p692] to the following:
            1. If *scripts*[0] is not *el*, then abort these steps.
            2. While *scripts* is not empty, and *scripts*[0]'s result[p692] is not "`uninitialized`":

1. Execute the script element[p698] *scripts*[0].

2. Remove *scripts*[0].

4. Otherwise, if *el* has a `defer`[p687] attribute or *el*'s type[p692] is "`module`":

    1. Append *el* to its parser document[p692]'s list of scripts that will execute when the document has finished parsing[p698].

    2. Set *el*'s steps to run when the result is ready[p692] to the following: set *el*'s ready to be parser-executed[p692] to true. (The parser will handle executing the script.)

5. Otherwise:

    1. Set *el*'s parser document[p692]'s pending parsing-blocking script[p698] to *el*.

    2. Block rendering[p142] on *el*.

    3. Set *el*'s steps to run when the result is ready[p692] to the following: set *el*'s ready to be parser-executed[p692] to true. (The parser will handle executing the script.)

36. Otherwise:

    1. Assert: *el*'s result[p692] is *not* "`uninitialized`".

    2. If all of the following are true:

        - *el*'s type[p692] is "`classic`";
        - *el* is parser-inserted[p692];
        - *el*'s parser document[p692] has a style sheet that is blocking scripts[p212]; and
        - either the parser that created *el* is an XML parser[p1484], or it's an HTML parser[p1368] whose script nesting level[p1370] is not greater than one,

        then:

        1. Set *el*'s parser document[p692]'s pending parsing-blocking script[p698] to *el*.

        2. Set *el*'s ready to be parser-executed[p692] to true. (The parser will handle executing the script.)

    3. Otherwise, immediately[p44] execute the script element[p698] *el*, even if other scripts are already executing.

Each `Document`[p135] has a **pending parsing-blocking script**, which is a `script`[p685] element or null, initially null.

Each `Document`[p135] has a **set of scripts that will execute as soon as possible**, which is a set of `script`[p685] elements, initially empty.

Each `Document`[p135] has a **list of scripts that will execute in order as soon as possible**, which is a list of `script`[p685] elements, initially empty.

Each `Document`[p135] has a **list of scripts that will execute when the document has finished parsing**, which is a list of `script`[p685] elements, initially empty.

Note

If a `script`[p685] element that blocks a parser gets moved to another `Document`[p135] before it would normally have stopped blocking that parser, it nonetheless continues blocking that parser until the condition that causes it to be blocking the parser no longer applies (e.g., if the script is a pending parsing-blocking script[p698] because the original `Document`[p135] has a style sheet that is blocking scripts[p212] when it was parsed, but then the script is moved to another `Document`[p135] before the blocking style sheet(s) loaded, the script still blocks the parser until the style sheets are all loaded, at which time the script executes and the parser is unblocked).

To **execute the script element** given a `script`[p685] element *el*:

1. Let *document* be *el*'s node document.

2. If *el*'s preparation-time document[p692] is not equal to *document*, then return.

3. Unblock rendering[p142] on *el*.
4. If *el*'s result[p692] is null, then fire an event named `error`[p1575] at *el*, and return.
5. If *el*'s from an external file[p692] is true, or *el*'s type[p692] is "`module`", then increment *document*'s ignore-destructive-writes counter[p1222].
6. Switch on *el*'s type[p692]:

   ↪ **"`classic`"**

   1. Let *oldCurrentScript* be the value to which *document*'s `currentScript`[p145] object was most recently set.
   2. If *el*'s root is *not* a shadow root, then set *document*'s `currentScript`[p145] attribute to *el*. Otherwise, set it to null.

      Note

      This does not use the in a document tree check, as el *could have been removed from the document prior to execution, and in that scenario* `currentScript`[p145] *still needs to point to it.*

   3. Run the classic script[p1165] given by *el*'s result[p692].
   4. Set *document*'s `currentScript`[p145] attribute to *oldCurrentScript*.

   ↪ **"`module`"**

   1. Assert: *document*'s `currentScript`[p145] attribute is null.
   2. Run the module script[p1165] given by *el*'s result[p692].

   ↪ **"`importmap`"**

   1. Register an import map[p1170] given *el*'s relevant global object[p1151] and *el*'s result[p692].

   ↪ **"`speculationrules`"**

   1. Register speculation rules[p1171] given *el*'s relevant global object[p1151] and *el*'s result[p692].

7. Decrement the ignore-destructive-writes counter[p1222] of *document*, if it was incremented in the earlier step.
8. If *el*'s from an external file[p692] is true, then fire an event named `load`[p1575] at *el*.

4.12.1.2 Scripting languages §[p699]

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`[p685] element.

Servers should use `text/javascript`[p1578] for JavaScript resources, in accordance with *Updates to ECMAScript Media Types*. Servers should not use other JavaScript MIME types for JavaScript resources, and must not use non-JavaScript MIME types. [RFC9239][p1586]

For external JavaScript resources, MIME type parameters in `Content-Type`[p102] headers are generally ignored. (In some cases the `charset` parameter has an effect.) However, for the `script`[p685] element's `type`[p686] attribute they are significant; it uses the JavaScript MIME type essence match concept.

Note

For example, scripts with their `type`[p686] *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`[p102] header processing as detailed in the prepare the script element[p694] algorithm and *Fetch*. [FETCH][p1582]

4.12.1.3 Restrictions for contents of `script` elements §[p70]

Note

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 "\x3C!--", "<script" as "\x3Cscript", and "</script" as "\x3C/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`[p685] blocks in HTML is a strange and exotic practice that acts unintuitively in the face of these sequences.

The `script`[p685] element's descendant text content must match the `script` production in the following ABNF, the character set for which is Unicode. [ABNF][p1579]

```
script        = outer *( comment-open inner comment-close outer )

outer         = < any string that doesn't contain a substring that matches not-in-outer >
not-in-outer  = comment-open
inner         = < any string that doesn't contain a substring that matches not-in-inner >
not-in-inner  = comment-close / script-open

comment-open  = "<!--"
comment-close = "-->"
script-open   = "<" s c r i p t tag-end

s             =  %x0053 ; U+0053 LATIN CAPITAL LETTER S
s             =/ %x0073 ; U+0073 LATIN SMALL LETTER S
c             =  %x0043 ; U+0043 LATIN CAPITAL LETTER C
c             =/ %x0063 ; U+0063 LATIN SMALL LETTER C
r             =  %x0052 ; U+0052 LATIN CAPITAL LETTER R
r             =/ %x0072 ; U+0072 LATIN SMALL LETTER R
i             =  %x0049 ; U+0049 LATIN CAPITAL LETTER I
i             =/ %x0069 ; U+0069 LATIN SMALL LETTER I
p             =  %x0050 ; U+0050 LATIN CAPITAL LETTER P
p             =/ %x0070 ; U+0070 LATIN SMALL LETTER P
t             =  %x0054 ; U+0054 LATIN CAPITAL LETTER T
t             =/ %x0074 ; U+0074 LATIN SMALL LETTER T

tag-end       =  %x0009 ; U+0009 CHARACTER TABULATION (tab)
tag-end       =/ %x000A ; U+000A LINE FEED (LF)
tag-end       =/ %x000C ; U+000C FORM FEED (FF)
tag-end       =/ %x0020 ; U+0020 SPACE
tag-end       =/ %x002F ; U+002F SOLIDUS (/)
tag-end       =/ %x003E ; U+003E GREATER-THAN SIGN (>)
```

When a `script`[p685] element contains script documentation[p701], 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:

```
const example = 'Consider this string: <!-- <script>';
console.log(example);
```

If one were to put this string directly in a `script`[p685] block, it would violate the restrictions above:

```
<script>
  const 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[p685] 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:

```
<script>
  const 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[p685] elements in HTML need to be balanced in order for the parser to consider closing the block.

By escaping the problematic strings as mentioned at the top of this section, the problem is avoided entirely:

```
<script>
  // Note: `\x3C` is an escape sequence for `<`.
  const example = 'Consider this string: \x3C!-- \x3Cscript>';
  console.log(example);
</script>
<!-- this is just a comment between script blocks -->
<script>
 ... // this is a new script block
</script>
```

It is possible for these sequences to naturally occur in script expressions, as in the following examples:

```
if (x<!--y) { ... }
if ( player<script ) { ... }
```

In such cases the characters cannot be escaped, but the expressions can be rewritten so that the sequences don't occur, as in:

```
if (x < !--y) { ... }
if (!--y > x) { ... }
if (!(--y) > x) { ... }
if (player < script) { ... }
if (script > player) { ... }
```

Doing this also avoids a different pitfall as well: for related historical reasons, the string "<!--" in classic scripts[p1153] is actually treated as a line comment start, just like "//".

4.12.1.4 Inline documentation for external scripts §[p70 1]

If a script[p685] element's src[p686] attribute is specified, then the contents of the script[p685] element, if any, must be such that the value of the text[p689] 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][p1579]

```
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[p685] 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[p686] attribute.

```
<script src="cool-effects.js">
 // create new instances using:
 //    var e = new Effect();
 // start the effect using .play, stop using .stop:
 //    e.play();
 //    e.stop();
</script>
```

4.12.1.5 Interaction of script[p685] elements and XSLT §[p702]

Avoid using client-side XSLT in browsers. The browser implementations of XSLT are highly susceptible to memory safety vulnerabilities, and the web platform has evolved toward JavaScript as the safe replacement. For these reasons, this feature is in the process of being removed from the web platform. (This is a long process that takes many years.)

This section is non-normative.

This specification does not define how XSLT interacts with the script[p685] 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[p685] elements created by the XSLT processor need to have its parser document[p692] set correctly, and run in document order (modulo scripts marked defer[p687] or async[p687]), immediately[p44], as the transformation is occurring.

- The XSLTProcessor[p53] transformToDocument()[p53] method adds elements to a Document[p135] object with a null browsing context[p1046], and, accordingly, any script[p685] elements they create need to have their already started[p692] set to true in the prepare the script element[p694] algorithm and never get executed (scripting is disabled[p1152]). Such script[p685] elements still need to have their parser document[p692] set, though, such that their async[p688] IDL attribute will return false in the absence of an async[p687] content attribute.

- The XSLTProcessor[p53] transformToFragment()[p53] 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[p685] elements with null parser document[p692] and with their already started[p692] set to false, 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[p135]s and the last operates on a fragment.

4.12.2 The noscript element §[p703]

Categories[p154]:

Metadata content[p156].
Flow content[p157].
Phrasing content[p157].

Contexts in which this element can be used[p154]:

In a head[p180] element of an HTML document, if there are no ancestor noscript[p703] elements.
Where phrasing content[p157] is expected in HTML documents, if there are no ancestor noscript[p703] elements.
As a descendant of a select[p590] element or an optgroup[p600] element, if there are no ancestor noscript[p703] elements.

Content model[p154]:

When scripting is disabled[p1152], in a head[p180] element: in any order, zero or more link[p184] elements, zero or more style[p207] elements, and zero or more meta[p196] elements.
When scripting is disabled[p1152], not in a head[p180] element: transparent[p159], but there must be no noscript[p703] element descendants.
Otherwise: text that conforms to the requirements given in the prose.

Tag omission in text/html[p154]:

Neither tag is omissible.

Content attributes[p154]:

Global attributes[p162]

Accessibility considerations[p154]:

For authors.
For implementers.

Sanitization[p154]:

Uncategorized[p1249].

DOM interface[p154]:

Uses HTMLElement[p149].

The noscript[p703] element represents[p148] nothing if scripting is enabled[p1152], and represents[p148] its children if scripting is disabled[p1152]. 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[p180] element, if scripting is disabled[p1152] for the noscript[p703] element

The noscript[p703] element must contain only link[p184], style[p207], and meta[p196] elements.

In a head[p180] element, if scripting is enabled[p1152] for the noscript[p703] element

The noscript[p703] element must contain only text, except that invoking the HTML fragment parsing algorithm[p1473] with the noscript[p703] element as the *context*[p1473] element and the text contents as the *input* must result in a list of nodes that consists only of link[p184], style[p207], and meta[p196] elements that would be conforming if they were children of the noscript[p703] element, and no parse errors[p1370].

Outside of head[p180] elements, if scripting is disabled[p1152] for the noscript[p703] element

The noscript[p703] element's content model is transparent[p159], with the additional restriction that a noscript[p703] element must not have a noscript[p703] element as an ancestor (that is, noscript[p703] can't be nested).

Outside of head[p180] elements, if scripting is enabled[p1152] for the noscript[p703] element

The noscript[p703] 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[p703] elements and no script[p685] elements, and such that no step in the algorithm throws an exception or causes an HTML parser[p1368] to flag a parse error[p1370]:

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.)

Note

All these contortions are required because, for historical reasons, the noscript element is handled differently by the HTML parser based on whether scripting mode was Disabled 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.

```
<form action="calcSquare.php">
 <p>
  <label for=x>Number</label>:
  <input id="x" name="x" type="number">
 </p>
 <script>
  var x = document.getElementById('x');
  var output = document.createElement('p');
  output.textContent = 'Type a number; it will be squared right then!';
  x.form.appendChild(output);
  x.form.onsubmit = function () { return false; }
  x.oninput = function () {
    var v = x.valueAsNumber;
    output.textContent = v + ' squared is ' + v * v;
  };
 </script>
 <noscript>
  <input type=submit value="Calculate Square">
 </noscript>
</form>
```

When script is disabled, a button appears to do the calculation on the server side. When script is enabled, the value is computed on-the-fly instead.

The noscript element is a blunt instrument. Sometimes, scripts might be enabled, but for some reason the page's script might fail. For this reason, it's generally better to avoid using noscript, and to instead design the script to change the page from being a scriptless page to a scripted page on the fly, as in the next example:

```
<form action="calcSquare.php">
 <p>
  <label for=x>Number</label>:
  <input id="x" name="x" type="number">
 </p>
```

```
<input id="submit" type=submit value="Calculate Square">
<script>
 var x = document.getElementById('x');
 var output = document.createElement('p');
 output.textContent = 'Type a number; it will be squared right then!';
 x.form.appendChild(output);
 x.form.onsubmit = function () { return false; }
 x.oninput = function () {
   var v = x.valueAsNumber;
   output.textContent = v + ' squared is ' + v * v;
 };
 var submit = document.getElementById('submit');
 submit.parentNode.removeChild(submit);
</script>
</form>
```

The above technique is also useful in 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
`for` — Updates existing content
`shadowrootmode` — Enables streaming declarative shadow roots
`shadowrootdelegatesfocus` — Sets delegates focus on a declarative shadow root
`shadowrootserializable` — Sets serializable on a declarative shadow root
`shadowrootslotassignment` — Sets slot assignment on a declarative shadow root
`shadowrootclonable` — Sets clonable on a declarative shadow root
`shadowrootcustomelementregistry` — Enables declarative shadow roots to indicate they will use a custom element registry

Accessibility considerations:

For authors.
For implementers.

Sanitization:

Uncategorized.

DOM interface:

IDL
```
[Exposed=Window]
interface HTMLTemplateElement : HTMLElement {
```

```
  [HTMLConstructor] constructor();

  readonly attribute DocumentFragment content;
  [CEReactions, Reflect="for"] attribute DOMString htmlFor;
  [CEReactions] attribute DOMString shadowRootMode;
  [CEReactions, Reflect] attribute boolean shadowRootDelegatesFocus;
  [CEReactions, Reflect] attribute boolean shadowRootSerializable;
  [CEReactions] attribute DOMString shadowRootSlotAssignment;
  [CEReactions, Reflect] attribute boolean shadowRootClonable;
  [CEReactions, Reflect] attribute DOMString shadowRootCustomElementRegistry;
};
```

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 **`for`** content attribute identifies the `ProcessingInstruction` that marks the location at which to insert content (`<?marker>`) or the start of the range to replace (`<?start>`). If the attribute is specified, the attribute's value must equal the value of the `name` attribute of a `<?marker>` or `<?start>` `ProcessingInstruction` in the same tree as the `template` element.

The **`shadowrootmode`** content attribute is an enumerated attribute with the following keywords and states:

Keyword	State	Brief description
`open`	**Open**	The template element represents an open declarative shadow root.
`closed`	**Closed**	The template element represents a closed declarative shadow root.

The `shadowrootmode` attribute's *invalid value default* and *missing value default* are both the **None** state.

The **`shadowrootdelegatesfocus`** content attribute is a boolean attribute.

The **`shadowrootserializable`** content attribute is a boolean attribute.

The **`shadowrootslotassignment`** content attribute is an enumerated attribute with the following keywords and states:

Keyword	State	Brief description
`named`	**Named**	The declarative shadow root uses named slot assignment.
`manual`	**Manual**	The declarative shadow root uses manual slot assignment.

The `shadowrootslotassignment` attribute's *invalid value default* and *missing value default* are both the Named state.

The **`shadowrootclonable`** content attribute is a boolean attribute.

The **`shadowrootcustomelementregistry`** content attribute is a boolean attribute.

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.

¶ 6 **Example**

For example, consider the following document:

```
<!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>
</html>
```

The p[p238] element in the template[p705] is *not* a child of the template[p705] in the DOM; it is a child of the DocumentFragment returned by the template[p705] element's content[p708] IDL attribute.

If the script were to call appendChild() on the template[p705] element, that would add a child to the template[p705] element (as for any other element); however, doing so is a violation of the template[p705] element's content model.

For web developers (non-normative)

***template*.content**[p708]

Returns the template contents[p707] (a DocumentFragment).

Each template[p705] element has an associated DocumentFragment object that is its **template contents**. The template contents[p707] have no conformance requirements[p148]. When a template[p705] element is created, the user agent must run the following steps to establish the template contents[p707]:

1. Let *document* be the template[p705] element's node document's appropriate template contents owner document[p707].
2. Create a DocumentFragment object whose node document is *document* and host is the template[p705] element.
3. Set the template[p705] element's template contents[p707] to the newly created DocumentFragment object.

A Document[p135] *document*'s **appropriate template contents owner document** is the Document[p135] returned by the following algorithm:

1. If *document* is not a Document[p135] created by this algorithm:
    1. If *document* does not yet have an **associated inert template document**:
        1. Let *newDocument* be the result of creating a document that implements Document[p135], given *document*'s relevant realm[p1151]; its browsing context[p1046] is null. This is "a Document[p135] created by this algorithm" for the purposes of the step above.
        2. If *document* is an HTML document, then mark *newDocument* as an HTML document also.
        3. Set *document*'s associated inert template document[p707] to *newDocument*.
    2. Set *document* to *document*'s associated inert template document[p707].

    Note

    Each Document[p135] not created by this algorithm thus gets a single Document[p135] to act as its proxy for owning the template contents[p707] of all its template[p705] elements, so that they aren't in a browsing context[p1046] and thus remain inert (e.g. scripts do not run). Meanwhile, template[p705] elements inside Document[p135] objects that are created by this algorithm just reuse the same Document[p135] owner for their contents.
2. Return *document*.

Each template[p705] element has an associated Element-or-DocumentFragment-or-null **insertion target**, which is initially null.

Each template[p705] element has an associated ProcessingInstruction-or-null **insertion start marker** and **insertion end marker**, which are initially null.

The adopting steps (with *node* and *oldDocument* as parameters) for template[p705] elements are the following:

1. Let *document* be *node*'s node document's appropriate template contents owner document[p707].

   Note

   node's *node document* *is the* Document[p135] *object that* node *was just adopted* into.

2. Adopt *node*'s template contents[p707] (a DocumentFragment object) into *document*.

The **content** getter steps are:

1. Assert: this's template contents[p707] is not a ShadowRoot node.
2. Return this's template contents[p707].

The **shadowRootMode** IDL attribute must reflect[p108] the shadowrootmode[p706] content attribute, limited to only known values[p109].

The **shadowRootSlotAssignment** IDL attribute must reflect[p108] the shadowrootslotassignment[p706] content attribute, limited to only known values[p109].

Note

The shadowRootCustomElementRegistry[p706] IDL attribute intentionally does not have a boolean type so it can be extended.

The cloning steps for template[p705] elements given *node*, *copy*, and *subtree* are:

1. If *subtree* is false, then return.
2. For each *child* of *node*'s template contents[p707]'s children, in tree order: clone a node given *child* with *document* set to *copy*'s template contents[p707]'s node document, *subtree* set to true, and *parent* set to *copy*'s template contents[p707].

To **prepare content patching** given an Element or DocumentFragment *scope* and a template[p705] element *template*:

1. Assert: *template*'s for[p706] attribute is specified.
2. Let *markerName* be *template*'s for[p706] attribute's value.
3. If *markerName* is the empty string, then return false.
4. Let (*start*, *end*) be the result of find markers[p709] given *scope* and *markerName*.
5. If *start* is null, then return false.
6. Set *template*'s insertion target[p707] to *start*'s parent.
7. Set *template*'s insertion start marker[p707] to *start*.
8. Set *template*'s insertion end marker[p707] to *end*.
9. If *start* is *end*, then return true.
10. Let *removedNodes* be an empty list.
11. Let *currentNode* be *start*'s next sibling.
12. While *currentNode* is not null:
    1. If *currentNode* is *end*, then break.
    2. Append *currentNode* to *removedNodes*.
    3. Set *currentNode* to *currentNode*'s next sibling.
13. For each *node* of *removedNodes*:
    1. If *node*'s parent is not null, then remove *node*.

       Note

       The parent might have changed because removing a node can fire events like the pagehide[p1576] *event, and an event handler can mutate the DOM. The parent identity is not checked; as long as the node still has a parent, it*

is removed.

14. Return true.

To **find markers** given a node *scope* and a string *name*:

1. For each *descendant* of *scope*'s descendants, in tree order:
    1. If *descendant* is not a `ProcessingInstruction` node, then continue.
    2. If the result of getting a processing instruction attribute given *descendant* and "`name`" is not *name*, then continue.
    3. If *descendant*'s target is "`marker`", then return (*descendant*, *descendant*).
    4. If *descendant*'s target is "`start`":
        1. Let *start* be *descendant*.
        2. Let *nestingLevel* be 0.
        3. Let *sibling* be *start*'s next sibling.
        4. While *sibling* is not null:
            1. If *sibling* is a `ProcessingInstruction` node:
                1. If *sibling*'s target is "`start`", then increment *nestingLevel* by one.
                2. If *sibling*'s target is "`end`":
                    1. If *nestingLevel* is 0, then return (*start*, *sibling*).
                    2. Decrement *nestingLevel* by one.
            2. Set *sibling* to *sibling*'s next sibling.
        5. Return (*start*, null).

        Note

        No matching `<?end>` *was found, which is interpreted as a range from* start *to the end of its parent.*
2. Return (null, null).

Example

In this example, a script populates a table four-column with data from a data structure, using a `template`[p705] to provide the element structure instead of manually generating the structure from markup.

```
<!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>
   <th>Name <th>Color <th>Sex <th>Legs
 <tbody>
  <template id="row">
   <tr><td><td><td><td>
  </template>
```

```
</table>
<script>
 var template = document.querySelector('#row');
 for (var i = 0; i < data.length; i += 1) {
   var cat = data[i];
   var clone = template.content.cloneNode(true);
   var cells = clone.querySelectorAll('td');
   cells[0].textContent = cat.name;
   cells[1].textContent = cat.color;
   cells[2].textContent = cat.sex;
   cells[3].textContent = cat.legs;
   template.parentNode.appendChild(clone);
 }
</script>
```

This example uses `cloneNode()` on the `template`[p705]'s contents; it could equivalently have used `document.importNode()`, which does the same thing. The only difference between these two APIs is when the node document is updated: with `cloneNode()` it is updated when the nodes are appended with `appendChild()`, with `document.importNode()` it is updated when the nodes are cloned.

Example

In this example, a `template`[p705] element with a `for`[p706] attribute is used to populate a list of countries in a `select`[p590] element, after the main content of the page has been delivered.

```
<!DOCTYPE html>
<html lang="en">
 <head>
  <title>Out-of-order countries</title>
  <style>
   #countries:empty {
    opacity: 0;
   }
  </style>
 </head>
 <body>
  <select id=countries><?marker name="country-options"?></select>
  ...
  <template for="country-options"><option>Antigua</option>...</template>
 </body>
</html>
```

The `<?marker?>` processing instruction is replaced with the `option`[p601] elements from the later `template`[p705] element. The `select`[p590] element is transparent until it is populated.

Example

In this example, a `template`[p705] element with a `for`[p706] attribute is used to stream a document out-of-order, sending fast static content first and recommendations computed asynchronously as they become available on the server.

```
<!DOCTYPE html>
<html lang="en">
 <head>
  <title>Async recommendation engine</title>
  <style>
   aside {
    /* position the aside where it can grow without causing layout shift or obscuring other content
*/
   }
   aside:has(.placeholder) {
```

```
    /* style the placeholder content */
   }
  </style>
 </head>
 <body>
  <h1>How to train your cat</h1>
  <p>Set realistic goals and don't expect your cat to make a perfect cup of tea
  on the first try.</p>
  <aside>
    <p>Recommended reading:</p>
    <?start name="recommended">
    <ul class="placeholder">
      <li>...</li>
      <li>...</li>
      <li>...</li>
    </ul>
    <?end>
  </aside>
  <p>...</p>
  <p>Now sit back and enjoy your tea with your cat.</p>

  <!-- The server has a potentially long delay at this point. -->

  <template for="recommended">
    <ul>
      <li>Cat or butler, which is right for you?</li>
      <li>Rewarding good cat behavior</li>
      <li>Kettles considered harmful</li>
    </ul>
  </template>
 </body>
</html>
```

The content between `<?start name="recommended">` and `<?end>` is replaced with the template[p705] contents. Since the new content does not use `class="placeholder"`, the placeholder styling no longer applies.

4.12.3.1 Interaction of template[p705] elements with XSLT and XPath §[p71] 1

> Avoid using client-side XSLT in browsers. The browser implementations of XSLT are highly susceptible to memory safety vulnerabilities, and the web platform has evolved toward JavaScript as the safe replacement. For these reasons, this feature is in the process of being removed from the web platform. (This is a long process that takes many years.)

This section is non-normative.

This specification does not define how XSLT and XPath interact with the template[p705] 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[p1484] needs to act as if template[p705] elements contain as descendants their template contents[p707] for the purposes of the transform.
- An XSLT processor that outputs a DOM needs to ensure that nodes that would go into a template[p705] element are instead placed into the element's template contents[p707].
- XPath evaluation using the XPath DOM API when applied to a Document[p135] parsed using the HTML parser[p1368] or the XML parser[p1484] described in this specification needs to ignore template contents[p707].

4.12.4 The slot element §[p712]

Categories[p154]:

Flow content[p157].
Phrasing content[p157].

Contexts in which this element can be used[p154]:

Where phrasing content[p157] is expected.

Content model[p154]:

Transparent[p159]

Tag omission in text/html[p154]:

Neither tag is omissible.

Content attributes[p154]:

Global attributes[p162]
name[p712] — Name of shadow tree slot

Accessibility considerations[p154]:

For authors.
For implementers.

Sanitization[p154]:

Uncategorized[p1249].

DOM interface[p154]:

IDL

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

  [CEReactions, Reflect] attribute DOMString name;
  sequence<Node> assignedNodes(optional AssignedNodesOptions options = {});
  sequence<Element> assignedElements(optional AssignedNodesOptions options = {});
  undefined assign((Element or Text)... nodes);
};

dictionary AssignedNodesOptions {
  boolean flatten = false;
};
```

The slot[p712] element defines a slot. It is typically used in a shadow tree. A slot[p712] element represents[p148] 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[p712] attribute is used to assign slots to other elements: a slot[p712] element with a name[p712] attribute creates a named slot to which any element is assigned if that element has a slot[p162] attribute whose value matches that name[p712] attribute's value, and the slot[p712] element is a child of the shadow tree whose root's host has that corresponding slot[p162] attribute value.

For web developers (non-normative)

***slot*.name[p712]**

Can be used to get and set *slot*'s name.

***slot*.assignedNodes[p713]()**

Returns *slot*'s assigned nodes.

***slot*.assignedNodes[p713]({ flatten: true })**

Returns *slot*'s assigned nodes, if any, and *slot*'s children otherwise, and does the same for any slot[p712] elements encountered therein, recursively, until there are no slot[p712] 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.

***slot*.assign(...*nodes*)**

Sets *slot*'s manually assigned nodes to the given *nodes*.

The `slot` element has **manually assigned nodes**, which is an ordered set of slottables set by `assign()`. This set is initially empty.

Note

The manually assigned nodes set can be implemented using weak references to the slottables, because this set is not directly accessible from script.

The **`assignedNodes(options)`** method steps are:

1. If *options*["`flatten`"] is false, then return this's assigned nodes.
2. Return the result of finding flattened slottables with this.

The **`assignedElements(options)`** method steps are:

1. If *options*["`flatten`"] is false, then return this's assigned nodes, filtered to contain only `Element` nodes.
2. Return the result of finding flattened slottables with this, filtered to contain only `Element` nodes.

The **`assign(...nodes)`** method steps are:

1. For each *node* of this's manually assigned nodes, set *node*'s manual slot assignment to null.
2. Let *nodesSet* be a new ordered set.
3. For each *node* of *nodes*:
    1. If *node*'s manual slot assignment refers to a slot, then remove *node* from that slot's manually assigned nodes.
    2. Set *node*'s manual slot assignment to this.
    3. Append *node* to *nodesSet*.
4. Set this's manually assigned nodes to *nodesSet*.
5. Run assign slottables for a tree for this's root.

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` elements, `img` elements with `usemap` attributes, `button` elements, `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[p154]:

Neither tag is omissible.

Content attributes[p154]:

Global attributes[p162]
width[p715] — Horizontal dimension
height[p715] — Vertical dimension

Accessibility considerations[p154]:

For authors.
For implementers.

Sanitization[p154]:

Uncategorized[p1249].

DOM interface[p154]:

IDL

```
typedef (CanvasRenderingContext2D or ImageBitmapRenderingContext or WebGLRenderingContext or
WebGL2RenderingContext or GPUCanvasContext) RenderingContext;

[Exposed=Window]
interface HTMLCanvasElement : HTMLElement {
  [HTMLConstructor] constructor();

  [CEReactions] attribute unsigned long width;
  [CEReactions] 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[p713] 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[p713] element in a document when a more suitable element is available. For example, it is inappropriate to use a canvas[p713] 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[p224]) and then styled using CSS and supporting technologies such as shadow trees.

When authors use the canvas[p713] element, they must also provide content that, when presented to the user, conveys essentially the same function or purpose as the canvas[p713]'s bitmap. This content may be placed as content of the canvas[p713] element. The contents of the canvas[p713] element, if any, are the element's fallback content[p158].

In interactive visual media, if scripting is enabled[p1152] for the canvas[p713] element, and if support for canvas[p713] elements has been enabled, then the canvas[p713] element represents[p148] embedded content[p158] consisting of a dynamically created image, the element's bitmap.

In non-interactive, static, visual media, if the canvas[p713] 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[p713] element represents[p148] embedded content[p158] with the element's current bitmap and size. Otherwise, the element represents its fallback content[p158] instead.

In non-visual media, and in visual media if scripting is disabled[p1152] for the canvas[p713] element or if support for canvas[p713] elements has been disabled, the canvas[p713] element represents[p148] its fallback content[p158] instead.

When a canvas element represents embedded content, the user can still focus descendants of the canvas element (in the fallback content). When an element is focused, it is the target of keyboard interaction events (even though the element itself is not visible). This allows authors to make an interactive canvas keyboard-accessible: authors should have a one-to-one mapping of interactive regions to *focusable areas* in the fallback content. (Focus has no effect on mouse interaction events.) [UIEVENTS]

An element whose nearest canvas element ancestor is being rendered and represents embedded content is an element that is **being used as relevant canvas fallback content**.

The canvas element has two attributes to control the size of the element's bitmap: **width** and **height**. These attributes, when specified, must have values that are valid non-negative integers. The rules for parsing non-negative integers must be used to **obtain their numeric values**. If an attribute is missing, or if parsing its value returns an error, then the default value must be used instead. The width attribute defaults to 300, and the height attribute defaults to 150.

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 natural dimensions of the canvas element when it represents embedded content are equal to the dimensions of the element's bitmap.

The user agent must use a square pixel density consisting of one pixel of image data per coordinate space unit for the bitmaps of a canvas and its rendering contexts.

Note

A canvas element can be sized arbitrarily by a style sheet, its bitmap is then subject to the 'object-fit' CSS property.

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, OffscreenCanvasRenderingContext2D, 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 also has a **canvas context mode**, which is initially **none** but can be changed to either **placeholder**, **2d**, **bitmaprenderer**, **webgl**, **webgl2**, or **webgpu** by algorithms defined in this specification.

When its canvas context mode is none, a canvas element has no rendering context, and its bitmap must be transparent black with a natural width equal to the numeric value of the element's width attribute and a natural 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 canvas 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 the OffscreenCanvas object's rendering context.

When a canvas element represents embedded content, it provides a paint source whose width is the element's natural width, whose height is the element's natural 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.

Context Mode	Action
2d	Follow the steps to set bitmap dimensions to the numeric values of the width and height content attributes.
webgl or **webgl2**	Follow the behavior defined in the WebGL specifications. [WEBGL]

Context Mode[p715]	Action
webgpu[p715]	Follow the behavior defined in *WebGPU*. [WEBGPU][p1588]
bitmaprenderer[p715]	If the context's bitmap mode[p775] is set to blank[p775], run the steps to set an ImageBitmapRenderingContext's output bitmap[p776], passing the canvas[p713] element's rendering context.
placeholder[p715]	Do nothing.
none[p715]	Do nothing.

The **width** and **height** IDL attributes must reflect[p108] the respective content attributes of the same name, with the same defaults.

For web developers (non-normative)

***context* = *canvas*.getContext[p716](*contextId* [, *options*])**

Returns an object that exposes an API for drawing on the canvas. *contextId* specifies the desired API: "2d[p716]", "bitmaprenderer[p716]", "webgl[p716]", "webgl2[p716]", or "webgpu[p717]". *options* is handled by that API.

This specification defines the "2d[p716]" and "bitmaprenderer[p716]" contexts below. The WebGL specifications define the "webgl[p716]" and "webgl2[p716]" contexts. *WebGPU* defines the "webgpu[p717]" context. [WEBGL][p1588] [WEBGPU][p1588]

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[p716]" context after getting a "webgl[p716]" context).

The **getContext(*contextId*, *options*)** method of the canvas[p713] 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[p713] element's canvas context mode[p715] and whose row header matches *contextId*:

	none[p715]	2d[p715]	bitmaprenderer[p715]	webgl[p715] or webgl2[p715]	webgpu[p715]	placeholder[p715]
"2d"	1. Let *context* be the result of running the 2D context creation algorithm[p724] given this and *options*. 2. Set this's context mode[p715] to 2d[p715]. 3. Return *context*.	Return the same object as was returned the last time the method was invoked with this same first argument.	Return null.	Return null.	Return null.	Throw an "InvalidStateError" DOMException.
"bitmaprenderer"	1. Let *context* be the result of running the ImageBitmapRenderingContext creation algorithm[p776] given this and *options*. 2. Set this's context mode[p715] to bitmaprenderer[p715]. 3. Return *context*.	Return null.	Return the same object as was returned the last time the method was invoked with this same first argument.	Return null.	Return null.	Throw an "InvalidStateError" DOMException.
"webgl" or "webgl2", if the user agent	1. Let *context* be the result of	Return null.	Return null.	Return the same object as	Return null.	Throw an "InvalidStateError" DOMException.

	none[p715]	2d[p715]	bitmaprenderer[p715]	webgl[p715] or webgl2[p715]	webgpu[p715]	placeholder[p715]
supports the WebGL feature in its current configuration	following the instructions given in the WebGL specifications' *Context Creation* sections. [WEBGL][p1588] 2. If *context* is null, then return null; otherwise set this's context mode[p715] to webgl[p715] or webgl2[p715]. 3. Return *context*.			was returned the last time the method was invoked with this same first argument.		
"webgpu", if the user agent supports the WebGPU feature in its current configuration	1. Let *context* be the result of following the instructions given in *WebGPU*'s Canvas Rendering section. [WEBGPU][p1588] 2. If *context* is null, then return null; otherwise set this's context mode[p715] to webgpu[p715]. 3. Return *context*.	Return null.	Return null.	Return null.	Return the same object as was returned the last time the method was invoked with this same first argument.	Throw an "InvalidStateError" DOMException.
An unsupported value*	Return null.	Return null.	Return null.	Return null.	Return null.	Throw an "InvalidStateError" DOMException.

* For example, the "webgl[p716]" or "webgl2[p716]" 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)

***url* = *canvas*.toDataURL[p717]([*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[p1577]"; 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[p1577]"), 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[p1577]", authors can check if the image was really returned in the requested format by checking to see if the returned string starts with one of the exact strings "data:image/png," or "data:image/png;". If it does, the image is PNG, and thus the requested type was not supported. (The one exception to this is if the canvas has either no height or no width, in which case the result might simply be "data:,".)

***canvas*.toBlob[p718](*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[p1577]"; 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[p1577]"), and is a number in the range 0.0 to 1.0 inclusive indicating the desired quality level for the resulting image.

***canvas*.transferControlToOffscreen[p718]()**

Returns a newly created OffscreenCanvas[p777] object that uses the canvas[p713] element as a placeholder. Once the canvas[p713] element has become a placeholder for an OffscreenCanvas[p777] object, its natural size can no longer be changed, and it cannot have a rendering context. The content of the placeholder canvas is updated on the OffscreenCanvas[p777]'s relevant agent[p1141]'s event loop[p1193]'s update the rendering[p1198] steps.

The **toDataURL(*type*, *quality*)** method, when invoked, must run these steps:

1. If this canvas[p713] element's bitmap's origin-clean[p715] flag is set to false, then throw a "SecurityError" DOMException.
2. If this canvas[p713] 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[p782], passing *type* and *quality* if given.
4. If *file* is null, then return "data:,".
5. Return a data: URL representing *file*. [RFC2397][p1585]

The **toBlob(*callback*, *type*, *quality*)** method, when invoked, must run these steps:

1. If this canvas[p713] element's bitmap's origin-clean[p715] flag is set to false, then throw a "SecurityError" DOMException.
2. Let *result* be null.
3. If this canvas[p713] 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[p713] element's bitmap.
4. Run these steps in parallel[p44]:
    1. If *result* is non-null, then set *result* to a serialization of *result* as a file[p782] with *type* and *quality* if given.
    2. Queue an element task[p1195] on the **canvas blob serialization task source** given the canvas[p713] element to run these steps:
        1. If *result* is non-null, then set *result* to a new Blob object, created in the relevant realm[p1151] of this canvas[p713] element, representing *result*. [FILEAPI][p1582]
        2. Invoke *callback* with « *result* » and "report".

The **transferControlToOffscreen()** method, when invoked, must run these steps:

1. If this canvas[p713] element's context mode[p715] is not set to none[p715], throw an "InvalidStateError" DOMException.
2. Let *offscreenCanvas* be a new OffscreenCanvas[p777] object with its width and height equal to the values of the width[p715] and height[p715] content attributes of this canvas[p713] element.
3. Set the *offscreenCanvas*'s placeholder canvas element[p777] to a weak reference to this canvas[p713] element.
4. Set this canvas[p713] element's context mode[p715] to placeholder[p715].
5. Set the *offscreenCanvas*'s inherited language[p778] to the language[p166] of this canvas[p713] element.
6. Set the *offscreenCanvas*'s inherited direction[p778] to the directionality[p168] of this canvas[p713] element.
7. Return *offscreenCanvas*.

4.12.5.1 The 2D rendering context §[p718]

IDL

```
typedef (HTMLImageElement or
         SVGImageElement) HTMLOrSVGImageElement;

typedef (HTMLOrSVGImageElement or
         HTMLVideoElement or
         HTMLCanvasElement or
         ImageBitmap or
         OffscreenCanvas or
         VideoFrame) CanvasImageSource;

enum CanvasColorType { "unorm8", "float16" };

enum CanvasFillRule { "nonzero", "evenodd" };

dictionary CanvasRenderingContext2DSettings {
  boolean alpha = true;
  boolean desynchronized = false;
```

```
  PredefinedColorSpace colorSpace = "srgb";
  CanvasColorType colorType = "unorm8";
  boolean willReadFrequently = false;
};

enum ImageSmoothingQuality { "low", "medium", "high" };

[Exposed=Window]
interface CanvasRenderingContext2D {
  // back-reference to the canvas
  readonly attribute HTMLCanvasElement canvas;
};
CanvasRenderingContext2D includes CanvasSettings;
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;
CanvasRenderingContext2D includes CanvasText;
CanvasRenderingContext2D includes CanvasDrawImage;
CanvasRenderingContext2D includes CanvasImageData;
CanvasRenderingContext2D includes CanvasPathDrawingStyles;
CanvasRenderingContext2D includes CanvasTextDrawingStyles;
CanvasRenderingContext2D includes CanvasPath;

interface mixin CanvasSettings {
  // settings
  CanvasRenderingContext2DSettings getContextAttributes();
};

interface mixin CanvasState {
  // state
  undefined save(); // push state on state stack
  undefined restore(); // pop state stack and restore state
  undefined reset(); // reset the rendering context to its default state
  boolean isContextLost(); // return whether context is lost
};

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);
  CanvasGradient createConicGradient(double startAngle, double x, double y);
  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 {
  // filters
  attribute DOMString filter; // (default "none")
};

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);
};

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, optional ImageDataSettings
settings = {});
  ImageData createImageData(ImageData imageData);
  ImageData getImageData([EnforceRange] long sx, [EnforceRange] long sy, [EnforceRange] long sw,
[EnforceRange] long sh, optional ImageDataSettings settings = {});
  undefined putImageData(ImageData imageData, [EnforceRange] long dx, [EnforceRange] long dy);
  undefined putImageData(ImageData imageData, [EnforceRange] long dx, [EnforceRange] long dy,
[EnforceRange] long dirtyX, [EnforceRange] long dirtyY, [EnforceRange] long dirtyWidth, [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" };
enum CanvasFontKerning { "auto", "normal", "none" };
enum CanvasFontStretch { "ultra-condensed", "extra-condensed", "condensed", "semi-condensed", "normal",
"semi-expanded", "expanded", "extra-expanded", "ultra-expanded" };
enum CanvasFontVariantCaps { "normal", "small-caps", "all-small-caps", "petite-caps", "all-petite-caps",
"unicase", "titling-caps" };
enum CanvasTextRendering { "auto", "optimizeSpeed", "optimizeLegibility", "geometricPrecision" };

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 lang; // (default: "inherit")
  attribute DOMString font; // (default 10px sans-serif)
  attribute CanvasTextAlign textAlign; // (default: "start")
  attribute CanvasTextBaseline textBaseline; // (default: "alphabetic")
  attribute CanvasDirection direction; // (default: "inherit")
  attribute DOMString letterSpacing; // (default: "0px")
  attribute CanvasFontKerning fontKerning; // (default: "auto")
  attribute CanvasFontStretch fontStretch; // (default: "normal")
  attribute CanvasFontVariantCaps fontVariantCaps; // (default: "normal")
  attribute CanvasTextRendering textRendering; // (default: "auto")
  attribute DOMString wordSpacing; // (default: "0px")
};

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 cp1x, unrestricted double cp1y, unrestricted double cp2x,
unrestricted double cp2y, 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 roundRect(unrestricted double x, unrestricted double y, unrestricted double w, unrestricted
double h, optional (unrestricted double or DOMPointInit or sequence<(unrestricted double or
DOMPointInit)>) radii = 0);
  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);
};

[Exposed=(Window,Worker)]
interface CanvasPattern {
  // opaque object
  undefined setTransform(optional DOMMatrix2DInit transform = {});
};

[Exposed=(Window,Worker)]
interface TextMetrics {
  // x-direction
  readonly attribute double width; // advance width
  readonly attribute double actualBoundingBoxLeft;
  readonly attribute double actualBoundingBoxRight;

  // y-direction
  readonly attribute double fontBoundingBoxAscent;
```

```
  readonly attribute double fontBoundingBoxDescent;
  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;
};

[Exposed=(Window,Worker)]
interface Path2D {
  constructor(optional (Path2D or DOMString) path);

  undefined addPath(Path2D path, optional DOMMatrix2DInit transform = {});
};
Path2D includes CanvasPath;
```

Note

To maintain compatibility with existing web content, user agents need to enumerate methods defined in CanvasUserInterface[p721] immediately after the stroke()[p757] method on CanvasRenderingContext2D[p719] objects.

For web developers (non-normative)

***context* = *canvas*.getContext[p716]('2d' [, { [alpha[p726]: true] [, desynchronized[p726]: false] [, colorSpace[p726]: 'srgb'] [, willReadFrequently[p726]: false]}])**

Returns a CanvasRenderingContext2D[p719] object that is permanently bound to a particular canvas[p713] element.

If the alpha[p726] member is false, then the context is forced to always be opaque.

If the desynchronized[p726] member is true, then the context might be desynchronized[p726].

The colorSpace[p726] member specifies the color space[p726] of the rendering context.

The colorType[p726] member specifies the color type[p726] of the rendering context.

If the willReadFrequently[p726] member is true, then the context is marked for readback optimization[p726].

***context*.canvas[p724]**

Returns the canvas[p713] element.

***attributes* = *context*.getContextAttributes[p726]()**

Returns an object whose:

- alpha[p725] member is true if the context has an alpha component that is not 1.0; otherwise false.
- desynchronized[p726] member is true if the context can be desynchronized[p726].
- colorSpace[p726] member is a string indicating the context's color space[p726].
- colorType[p726] member is a string indicating the context's color type[p726].
- willReadFrequently[p726] member is true if the context is marked for readback optimization[p726].

The CanvasRenderingContext2D[p719] 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[p725] in CSS pixels; similarly, the *y*-coordinate of the bottom-most edge is equal to the height of the rendering context's output bitmap[p725] 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[p725] 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[p713] element below's rendering context has a 200x200 output bitmap[p725] (which internally uses CSS pixels as a unit for ease of integration with CSS) and is rendered as 100x100 CSS pixels:

```
<canvas width=200 height=200 style=width:100px;height:100px>
```

The **2D context creation algorithm**, which is passed a *target* (a canvas[p713] element) and *options*, consists of running these steps:

1. Let *settings* be the result of converting *options* to the dictionary type CanvasRenderingContext2DSettings[p718]. (This can throw an exception.)
2. Let *context* be a new CanvasRenderingContext2D[p719] object.
3. Initialize *context*'s canvas[p724] attribute to point to *target*.
4. Set *context*'s output bitmap[p725] to the same bitmap as *target*'s bitmap (so that they are shared).
5. Set bitmap dimensions[p724] to the numeric values[p715] of *target*'s width[p715] and height[p715] content attributes.
6. Run the canvas settings output bitmap initialization algorithm[p726], given *context* and *settings*.
7. 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[p727].
2. Resize the output bitmap[p725] to the new *width* and *height*.
3. Let *canvas* be the canvas[p713] element to which the rendering context's canvas[p724] attribute was initialized.
4. If the numeric value[p715] of *canvas*'s width[p715] content attribute differs from *width*, then set *canvas*'s width[p715] content attribute to the shortest possible string representing *width* as a valid non-negative integer[p81].
5. If the numeric value[p715] of *canvas*'s height[p715] content attribute differs from *height*, then set *canvas*'s height[p715] content attribute to the shortest possible string representing *height* as a valid non-negative integer[p81].

Example

Only one square appears to be drawn in the following example:

```
// canvas is a reference to a <canvas> element
var context = canvas.getContext('2d');
context.fillRect(0,0,50,50);
canvas.setAttribute('width', '300'); // clears the canvas
context.fillRect(0,100,50,50);
canvas.width = canvas.width; // clears the canvas
context.fillRect(100,0,50,50); // only this square remains
```

The **canvas** attribute must return the value it was initialized to when the object was created.

The CanvasColorType[p718] enumeration is used to specify the color type[p726] of the canvas's backing store.

The "**unorm8**" value indicates that the type for all color components is 8-bit unsigned normalized.

The "**float16**" value indicates that the type for all color components is 16-bit floating point.

The CanvasFillRule[p718] 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[p719] 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 §[p72 5]

This section is non-normative.

The output bitmap[p725], 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()[p760], or the createImageBitmap()[p1276] factory method). In many cases, this will be more memory efficient.

The bitmap of a canvas[p713] element is the one bitmap that's pretty much always going to be needed in practice. The output bitmap[p725] of a rendering context, when it has one, is always just an alias to a canvas[p713] 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 natural 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 settings §[p72 5]

A CanvasSettings[p719] object has an **output bitmap** that is initialized when the object is created.

The output bitmap[p725] has an origin-clean[p715] flag, which can be set to true or false. Initially, when one of these bitmaps is created, its origin-clean[p715] flag must be set to true.

The CanvasSettings[p719] object also has an **alpha** boolean. When a CanvasSettings[p719] object's alpha[p725] is false, then its alpha component 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.

Note

Thus, the bitmap of such a context starts off as opaque black instead of transparent black; clearRect()[p753] always results in opaque black pixels, every fourth byte from getImageData()[p763] is always 255, the putImageData()[p763] 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 component; 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 CanvasSettings object also has a **desynchronized** boolean. When a CanvasSettings 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 CanvasSettings object also has a **will read frequently** boolean. When a CanvasSettings object's will read frequently is true, the user agent may optimize the canvas for readback operations.

Note

On most devices the user agent needs to decide whether to store the canvas's output bitmap on the GPU (this is also called "hardware accelerated"), or on the CPU (also called "software"). Most rendering operations are more performant for accelerated canvases, with the major exception being readback with getImageData(), toDataURL(), or toBlob(). CanvasSettings objects with will read frequently equal to true tell the user agent that the webpage is likely to perform many readback operations and that it is advantageous to use a software canvas.

The CanvasSettings object also has a **color space** setting of type PredefinedColorSpace. The CanvasSettings object's color space indicates the color space for the output bitmap.

The CanvasSettings object also has a **color type** setting of type CanvasColorType. The CanvasSettings object's color type indicates the data type of the color and alpha components of the pixels of the output bitmap.

To **initialize a `CanvasSettings` output bitmap**, given a CanvasSettings *context* and a CanvasRenderingContext2DSettings *settings*:

1. Set *context*'s alpha to *settings*["**`alpha`**"].
2. Set *context*'s desynchronized to *settings*["**`desynchronized`**"].
3. Set *context*'s color space to *settings*["**`colorSpace`**"].
4. Set *context*'s color type to *settings*["**`colorType`**"].
5. Set *context*'s will read frequently to *settings*["**`willReadFrequently`**"].

The **`getContextAttributes()`** method steps are to return «["alpha" → this's alpha, "desynchronized" → this's desynchronized, "colorSpace" → this's color space, "colorType" → this's color type, "willReadFrequently" → this's will read frequently]».

4.12.5.1.3 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[p757].
- The current letter spacing[p734], word spacing[p734], fill style[p749], stroke style[p749], filter[p768], global alpha[p766], compositing and blending operator[p766], and shadow color[p767].
- The current values of the following attributes: lineWidth[p728], lineCap[p728], lineJoin[p728], miterLimit[p729], lineDashOffset[p729], shadowOffsetX[p767], shadowOffsetY[p767], shadowBlur[p768], lang[p734], font[p733], textAlign[p734], textBaseline[p734], direction[p734], fontKerning[p734], fontStretch[p735], fontVariantCaps[p735], textRendering[p735], imageSmoothingEnabled[p767], imageSmoothingQuality[p767].
- The current dash list[p729].

Note

The rendering context's bitmaps are not part of the drawing state, as they depend on whether and how the rendering context is bound to a canvas[p713] element.

Objects that implement the CanvasState[p719] mixin have a **context lost** boolean, that is initialized to false when the object is created. The context lost[p727] value is updated in the context lost steps[p1199].

For web developers (non-normative)

***context*.save[p727]()**

Pushes the current state onto the stack.

***context*.restore[p727]()**

Pops the top state on the stack, restoring the context to that state.

***context*.reset[p727]()**

Resets the rendering context, which includes the backing buffer, the drawing state stack, path, and styles.

***context*.isContextLost[p727]()**

Returns true if the rendering context was lost. Context loss can occur due to driver crashes, running out of memory, etc. In these cases, the canvas loses its backing storage and takes steps to reset the rendering context to its default state[p727].

The **save()** method steps are to push a copy of the current drawing state onto the drawing state stack.

The **restore()** method steps are to 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.

The **reset()** method steps are to reset the rendering context to its default state[p727].

To **reset the rendering context to its default state**:

1. Clear canvas's bitmap to transparent black.
2. Empty the list of subpaths in the context's current default path[p756].
3. Clear the context's drawing state stack.
4. Reset everything that drawing state[p726] consists of to their initial values.

The **isContextLost()** method steps are to return this's context lost[p727].

4.12.5.1.4 Line styles §[p727]

For web developers (non-normative)

***context*.lineWidth[p728] [= *value*]**

***styles*.lineWidth[p728] [= *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.

***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.

***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.

***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.

***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.

***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).

***context*.lineDashOffset**
***styles*.lineDashOffset**

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[p728] 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[p721] interface is created, the miterLimit[p729] attribute must initially have the value 10.0.

Each CanvasPathDrawingStyles[p721] object has a **dash list**, which is either empty or consists of an even number of non-negative numbers. Initially, the dash list[p729] must be empty.

The **setLineDash(*segments*)** method, when invoked, must run these steps:

1. If any value in *segments* 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).
2. If the number of elements in *segments* is odd, then let *segments* be the concatenation of two copies of *segments*.
3. Set the object's dash list[p729] to *segments*.

When the **getLineDash()** method is invoked, it must return a sequence whose values are the values of the object's dash list[p729], 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[p721] interface is created, the lineDashOffset[p729] 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[p721] interface, it must run the following algorithm. This algorithm returns a new path[p739].

1. Let *path* be a copy of the path being traced.
2. Prune all zero-length line segments[p739] from *path*.
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[p729] 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[p729], 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[p729], 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 0 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[p729]'s *index*th 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[p729]'s *index*th 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 *start* finds itself into two at *start* and place a point there, cutting in two the subpath that it was in, and similarly 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, and then remove all line segments, joins, points, and subpaths that are between *start* and *end*.

    If *start* and *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 *position* is greater than *subpath width*, then jump to the step labeled *convert*.
21. Increment *index* by one. If it is equal to the number of entries in *style*'s dash list[p729], then let *index* be 0.
22. 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 path[p739] that describes the edge of the areas that would be covered if a straight line of length equal to *style*'s `lineWidth`[p728] was swept along each subpath in *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 *style*'s `lineCap`[p728] attribute as described previously and elaborated below, and replacing each join with the join necessary to satisfy *style*'s `lineJoin`[p728] 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 *style*'s `lineCap`[p728]. The "`butt`" value means that no additional line cap is added. The "`round`" value means that a semi-circle with the diameter equal to *style*'s `lineWidth`[p728] 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 *style*'s lineWidth[p728] width and the width of half *style*'s lineWidth[p728] 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 lineJoin[p728] 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 *style*'s miterLimit[p729] 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.

4.12.5.1.5 Text styles §[p731]

For web developers (non-normative)

***context*.lang**[p734] **[= *value*]**

***styles*.lang**[p734] **[= *value*]**

Returns the current language setting.

Can be set to a BCP 47 language tag, the empty string, or "inherit[p734]", to change the language used when resolving fonts. "inherit[p734]" takes the language from the canvas[p713] element's language, or the associated document element when there is no canvas[p713] element.

The default is "inherit[p734]".

***context*.font**[p733] **[= *value*]**

***styles*.font**[p733] **[= *value*]**

Returns the current font settings.

Can be set, to change the font. The syntax is the same as for the CSS 'font' property; values that cannot be parsed as CSS font values are ignored. The default is "10px sans-serif".

Relative keywords and lengths are computed relative to the font of the canvas[p713] element.

***context*.textAlign**[p734] **[= *value*]**

***styles*.textAlign**[p734] **[= *value*]**

Returns the current text alignment settings.

Can be set, to change the alignment. The possible values are and their meanings are given below. Other values are ignored. The default is "start".

***context*.textBaseline**[p734] **[= *value*]**

***styles*.textBaseline**[p734] **[= *value*]**

Returns the current baseline alignment settings.

Can be set, to change the baseline alignment. The possible values and their meanings are given below. Other values are ignored. The default is "alphabetic"[p735].

***context*.direction**[p734] **[= *value*]**

***styles*.direction**[p734] **[= *value*]**

Returns the current directionality.

Can be set, to change the directionality. The possible values and their meanings are given below. Other values are ignored. The default is "inherit"[p736].

***context*.letterSpacing**[p734] **[= *value*]**

***styles*.letterSpacing**[p734] **[= *value*]**

Returns the current spacing between characters in the text.

Can be set, to change spacing between characters. Values that cannot be parsed as a CSS <length> are ignored. The default is "0px".

***context*.fontKerning**[p734] **[= *value*]**

***styles*.fontKerning**[p734] **[= *value*]**

Returns the current font kerning settings.

Can be set, to change the font kerning. The possible values and their meanings are given below. Other values are ignored. The default is "auto"[p736].

***context*.fontStretch**[p735] **[= *value*]**

***styles*.fontStretch**[p735] **[= *value*]**

Returns the current font stretch settings.

Can be set, to change the font stretch. The possible values and their meanings are given below. Other values are ignored. The default is "normal"[p736].

***context*.fontVariantCaps**[p735] **[= *value*]**

***styles*.fontVariantCaps**[p735] **[= *value*]**

Returns the current font variant caps settings.

Can be set, to change the font variant caps. The possible values and their meanings are given below. Other values are ignored. The default is "normal"[p736].

***context*.textRendering**[p735] **[= *value*]**

***styles*.textRendering**[p735] **[= *value*]**

Returns the current text rendering settings.

Can be set, to change the text rendering. The possible values and their meanings are given below. Other values are ignored. The default is "auto"[p737].

***context*.wordSpacing**[p734] **[= *value*]**

***styles*.wordSpacing**[p734] **[= *value*]**

Returns the current spacing between words in the text.

Can be set, to change spacing between words. Values that cannot be parsed as a CSS <length> are ignored. The default is "0px".

Objects that implement the CanvasTextDrawingStyles[p722] 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[p719] objects, this is the canvas[p713] element given by the value of the context's canvas[p724] attribute. For OffscreenCanvasRenderingContext2D[p781] objects, this is the associated OffscreenCanvas object[p781].

Font resolution for the font style source object[p732] requires a font source. This is determined for a given *object* implementing CanvasTextDrawingStyles[p722] by the following steps: [CSSFONTLOAD][p1580]

1. If *object*'s font style source object[p732] is a canvas[p713] element, return the element's node document.
2. Otherwise, *object*'s font style source object[p732] is an OffscreenCanvas[p777] object:

1. Let *global* be *object*'s relevant global object[p1151].
2. If *global* is a Window[p965] object, then return *global*'s associated Document[p966].
3. Assert: *global* implements WorkerGlobalScope[p1322].
4. Return *global*.

Example

This is an example of font resolution with a regular canvas[p713] element with ID `c1`.

```
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[p777]. Assuming a canvas[p713] element with ID `c2` which is transferred to a worker like so:

```
const offscreenCanvas = document.getElementById("c2").transferControlToOffscreen();
worker.postMessage(offscreenCanvas, [offscreenCanvas]);
```

Then, in the worker:

```
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][p1580]

Font family names must be interpreted in the context of the font style source object[p732] when the font is to be used; any fonts embedded using `@font-face` or loaded using FontFace[p69] objects that are visible to the font style source object[p732] must therefore be available once they are loaded. (Each font style source object[p732] 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[p732] 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][p1580] [CSSFONTLOAD][p1580]

On getting, the font[p733] attribute must return the serialized form of the current font of the context (with no 'line-height' component). [CSSOM][p1581]

Example

For example, after the following statement:

```
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[p722] 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[p732] 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[p732] 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[p732] is not an element or is not being rendered[p1488]), 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[p722] interface is created, the textAlign[p734] attribute must initially have the value start[p735].

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[p722] interface is created, the textBaseline[p734] attribute must initially have the value alphabetic[p735].

Objects that implement the CanvasTextDrawingStyles[p722] interface have an associated **language** value used to localize font rendering. Valid values are a BCP 47 language tag, the empty string, or "**inherit**" where the language comes from the canvas[p713] element's language, or the associated document element when there is no canvas[p713] element. Initially, the language[p734] must be "inherit[p734]".

The **lang** getter steps are to return this's language[p734].

The lang[p734] setter steps are to set this's language[p734] to the given value.

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[p722] interface is created, the direction[p734] attribute must initially have the value "inherit[p736]".

Objects that implement the CanvasTextDrawingStyles[p722] interface have attributes that control the spacing between letters and words. Such objects have associated **letter spacing** and **word spacing** values, which are CSS <length> values. Initially, both must be the result of parsing "`0px`" as a CSS <length>.

The **letterSpacing** getter steps are to return the serialized form of this's letter spacing[p734].

The letterSpacing[p734] setter steps are:

1. Let *parsed* be the result of parsing the given value as a CSS <length>.
2. If *parsed* is failure, then return.
3. Set this's letter spacing[p734] to *parsed*.

The **wordSpacing** getter steps are to return the serialized form of this's word spacing[p734].

The wordSpacing[p734] setter steps are:

1. Let *parsed* be the result of parsing the given value as a CSS <length>.
2. If *parsed* is failure, then return.
3. Set this's word spacing[p734] to *parsed*.

The **fontKerning** 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[p722] interface is created, the fontKerning[p734] attribute must initially have the value "auto[p736]".

The **fontStretch** 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[p722] interface is created, the fontStretch[p735] attribute must initially have the value "normal[p736]".

The **fontVariantCaps** 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[p722] interface is created, the fontVariantCaps[p735] attribute must initially have the value "normal[p736]".

The **textRendering** 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[p722] interface is created, the textRendering[p735] attribute must initially have the value "auto[p737]".

The textAlign[p734] 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[p734] attribute's allowed keywords correspond to alignment points in the font:

The keywords map to these alignment points as follows:

top
: The em-over baseline

hanging
: The hanging baseline

middle
: Halfway between the em-over baseline and the em-under baseline

alphabetic
: The alphabetic baseline

ideographic
The ideographic-under baseline

bottom
The em-under baseline

The `direction`[p734] attribute's allowed keywords are as follows:

ltr
Treat input to the text preparation algorithm[p737] as left-to-right text.

rtl
Treat input to the text preparation algorithm[p737] as right-to-left text.

inherit
Use the process in the text preparation algorithm[p737] to obtain the text direction from the `canvas`[p713] element, placeholder `canvas` element[p777], or document element.

The `fontKerning`[p734] attribute's allowed keywords are as follows:

auto
Kerning is applied at the discretion of the user agent.

normal
Kerning is applied.

none
Kerning is not applied.

The `fontStretch`[p735] attribute's allowed keywords are as follows:

ultra-condensed
Same as CSS 'font-stretch' 'ultra-condensed' setting.

extra-condensed
Same as CSS 'font-stretch' 'extra-condensed' setting.

condensed
Same as CSS 'font-stretch' 'condensed' setting.

semi-condensed
Same as CSS 'font-stretch' 'semi-condensed' setting.

normal
The default setting, where width of the glyphs is at 100%.

semi-expanded
Same as CSS 'font-stretch' 'semi-expanded' setting.

expanded
Same as CSS 'font-stretch' 'expanded' setting.

extra-expanded
Same as CSS 'font-stretch' 'extra-expanded' setting.

ultra-expanded
Same as CSS 'font-stretch' 'ultra-expanded' setting.

The `fontVariantCaps`[p735] attribute's allowed keywords are as follows:

normal
None of the features listed below are enabled.

small-caps
Same as CSS 'font-variant-caps' 'small-caps' setting.

all-small-caps
Same as CSS 'font-variant-caps' 'all-small-caps' setting.

petite-caps
Same as CSS 'font-variant-caps' 'petite-caps' setting.

all-petite-caps
Same as CSS 'font-variant-caps' 'all-petite-caps' setting.

unicase
Same as CSS 'font-variant-caps' 'unicase' setting.

titling-caps
Same as CSS 'font-variant-caps' 'titling-caps' setting.

The `textRendering`[p735] attribute's allowed keywords are as follows:

auto
Same as 'auto' in SVG text-rendering property.

optimizeSpeed
Same as 'optimizeSpeed' in SVG text-rendering property.

optimizeLegibility
Same as 'optimizeLegibility' in SVG text-rendering property.

geometricPrecision
Same as 'geometricPrecision' in SVG text-rendering property.

The **text preparation algorithm** is as follows. It takes as input a string *text*, a `CanvasTextDrawingStyles`[p722] 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`[p733] attribute.
4. Let *language* be the *target*'s language[p734].
5. If *language* is "`inherit`[p734]":
    1. Let *sourceObject* be *object*'s font style source object[p732].
    2. If *sourceObject* is a `canvas`[p713] element, then set *language* to the *sourceObject*'s language[p166].
    3. Otherwise:
        1. Assert: *sourceObject* is an `OffscreenCanvas`[p777] object.
        2. Set *language* to the *sourceObject*'s inherited language[p778].
6. If *language* is the empty string, then set *language* to explicitly unknown[p166].
7. Apply the appropriate step from the following list to determine the value of *direction*:

    ↪ **If the *target* object's `direction`[p734] attribute has the value "`ltr`[p736]"**
    Let *direction* be 'ltr[p168]'.

    ↪ **If the *target* object's `direction`[p734] attribute has the value "`rtl`[p736]"**
    Let *direction* be 'rtl[p168]'.

↪ **If the *target* object's `direction`[p734] attribute has the value "`inherit`[p736]"**

1. Let *sourceObject* be *object*'s font style source object[p732].
2. If *sourceObject* is a `canvas`[p713] element, then let *direction* be *sourceObject*'s directionality[p168].
3. Otherwise:
    1. Assert: *sourceObject* is an `OffscreenCanvas`[p777] object.
    2. Let *direction* be *sourceObject*'s inherited direction[p778].

8. Form a hypothetical infinitely-wide CSS line box containing a single inline box containing the text *text*, with the CSS content language set to *language*, and with its CSS properties set as follows:

Property	Source
'direction'	*direction*
'font'	*font*
'font-kerning'	*target*'s `fontKerning`[p734]
'font-stretch'	*target*'s `fontStretch`[p735]
'font-variant-caps'	*target*'s `fontVariantCaps`[p735]
'letter-spacing'	*target*'s letter spacing[p734]
SVG text-rendering	*target*'s `textRendering`[p735]
'white-space'	'pre'
'word-spacing'	*target*'s word spacing[p734]

and with all other properties set to their initial values.

9. 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.

10. 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`[p734] and `textBaseline`[p734] values as follows:

    Horizontal position:

    If `textAlign`[p734] is `left`[p735]
    If `textAlign`[p734] is `start`[p735] and *direction* is 'ltr'
    If `textAlign`[p734] is `end`[p735] and *direction* is 'rtl'
    Set the *anchor point*'s horizontal position to the left edge of the inline box, and let *physical alignment* be *left*.

    If `textAlign`[p734] is `right`[p735]
    If `textAlign`[p734] is `end`[p735] and *direction* is 'ltr'
    If `textAlign`[p734] is `start`[p735] and *direction* is 'rtl'
    Set the *anchor point*'s horizontal position to the right edge of the inline box, and let *physical alignment* be *right*.

    If `textAlign`[p734] is `center`[p735]
    Set the *anchor point*'s horizontal position to half way between the left and right edges of the inline box, and let *physical alignment* be *center*.

    Vertical position:

    If `textBaseline`[p734] is `top`[p735]
    Set the *anchor point*'s vertical position to the top of the em box of the first available font of the inline box.

    If `textBaseline`[p734] is `hanging`[p735]
    Set the *anchor point*'s vertical position to the hanging baseline of the first available font of the inline box.

    If `textBaseline`[p734] is `middle`[p735]
    Set the *anchor point*'s vertical position to half way between the bottom and the top of the em box of the first available font of the inline box.

    If `textBaseline`[p734] is `alphabetic`[p735]
    Set the *anchor point*'s vertical position to the alphabetic baseline of the first available font of the inline box.

If textBaseline[p734] is ideographic[p736]

Set the *anchor point*'s vertical position to the ideographic-under baseline of the first available font of the inline box.

If textBaseline[p734] is bottom[p736]

Set the *anchor point*'s vertical position to the bottom of the em box of the first available font of the inline box.

11. 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 at the *anchor point*.

12. Return *result*, *physical alignment*, and the inline box.

4.12.5.1.6 Building paths §[p739]

Objects that implement the CanvasPath[p722] interface have a path[p739]. 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[p739] 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[p739] is created, its need new subpath[p739] flag must be set.

When an object implementing the CanvasPath[p722] interface is created, its path[p739] must be initialized to zero subpaths.

For web developers (non-normative)

***context*.moveTo[p742](*x*, *y*)**
***path*.moveTo[p742](*x*, *y*)**

Creates a new subpath with the given point.

***context*.closePath[p742]()**
***path*.closePath[p742]()**

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[p742](*x*, *y*)**
***path*.lineTo[p742](*x*, *y*)**

Adds the given point to the current subpath, connected to the previous one by a straight line.

***context*.quadraticCurveTo[p742](*cpx*, *cpy*, *x*, *y*)**
***path*.quadraticCurveTo[p742](*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.

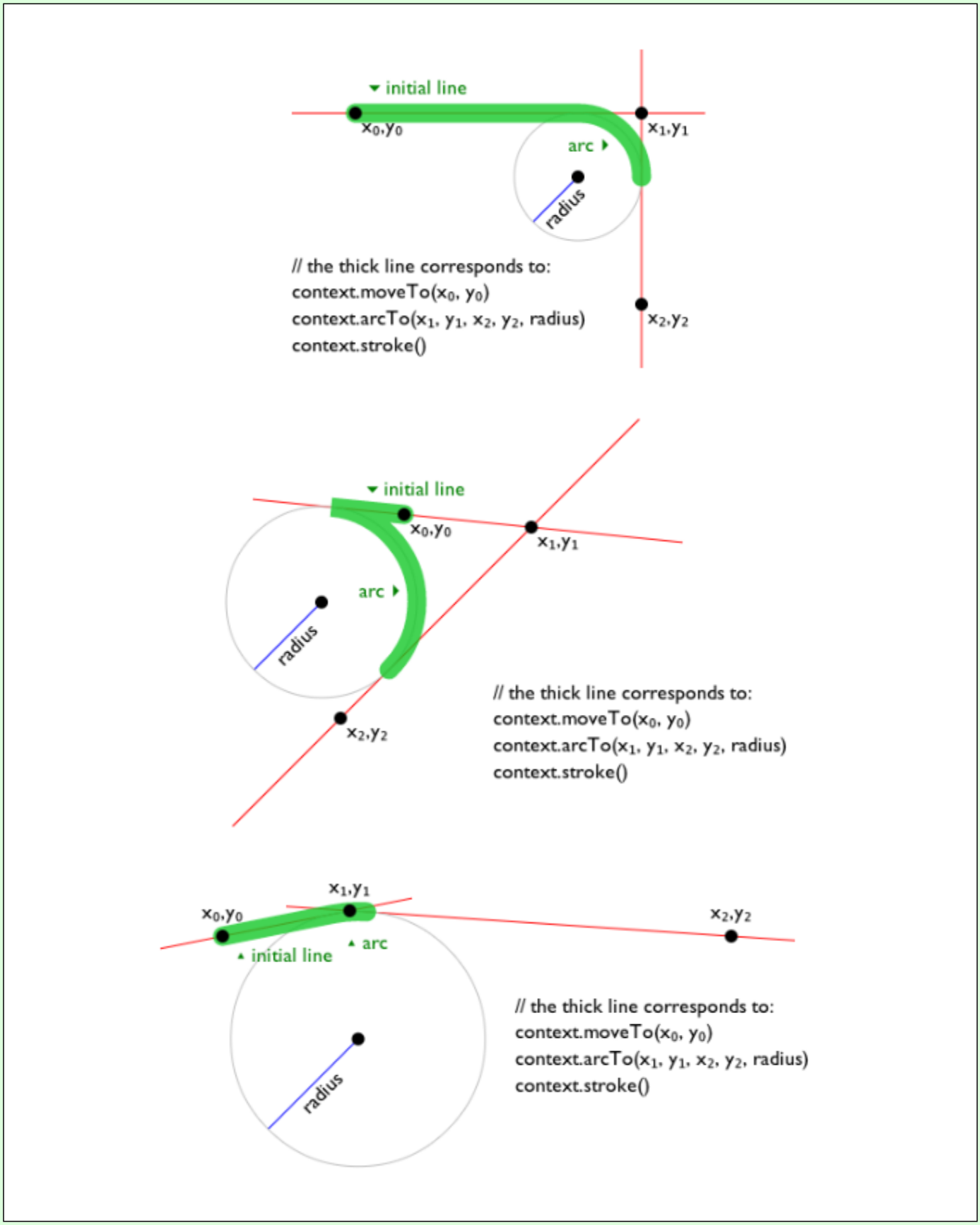
***context*.bezierCurveTo[p742](*cp1x*, *cp1y*, *cp2x*, *cp2y*, *x*, *y*)**
***path*.bezierCurveTo[p742](*cp1x*, *cp1y*, *cp2x*, *cp2y*, *x*, *y*)**

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[p742](*x1*, *y1*, *x2*, *y2*, *radius*)**
***path*.arcTo[p742](*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.



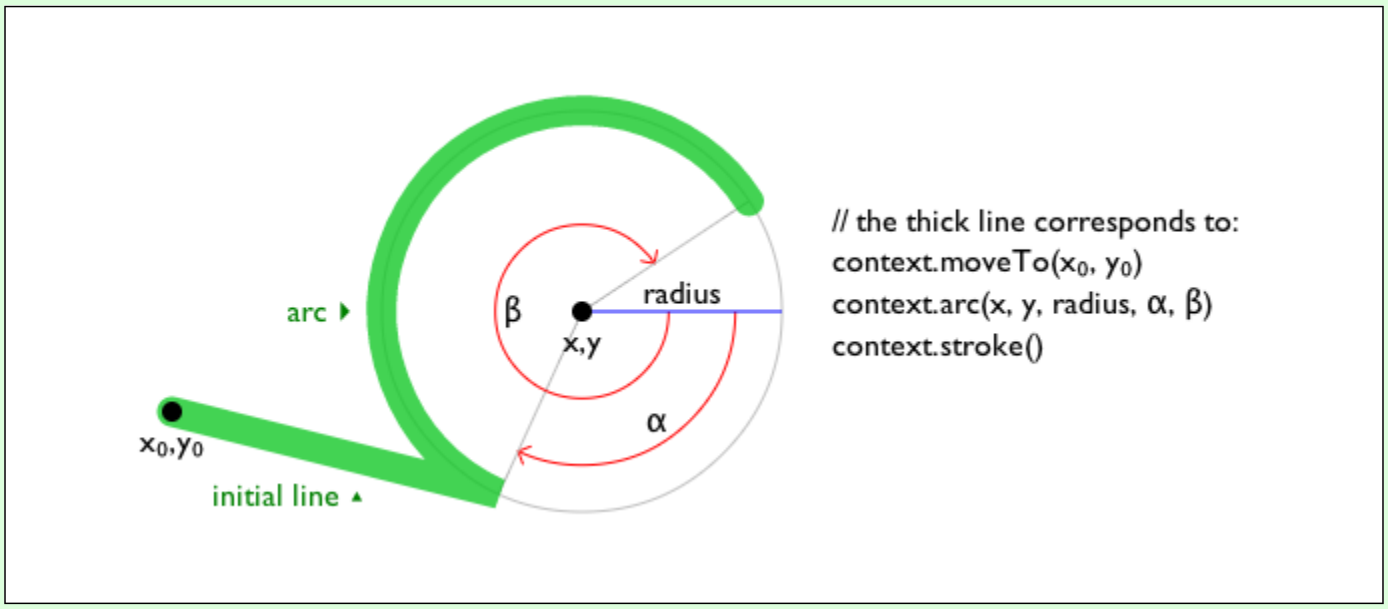
***context*.arc**[p743]**(*x*, *y*, *radius*, *startAngle*, *endAngle* [, *counterclockwise*])**

***path*.arc**[p743]**(*x*, *y*, *radius*, *startAngle*, *endAngle* [, *counterclockwise*])**

Adds points to the subpath such that the arc described by the circumference of the circle described by the arguments, starting at the given start angle and ending at the given end angle, going in the given direction (defaulting to clockwise), is added to the

path, connected to the previous point by a straight line.

Throws an "IndexSizeError" DOMException if the given radius is negative.



***context*.ellipse[p743](*x*, *y*, *radiusX*, *radiusY*, *rotation*, *startAngle*, *endAngle* [, *counterclockwise*])**

***path*.ellipse[p743](*x*, *y*, *radiusX*, *radiusY*, *rotation*, *startAngle*, *endAngle* [, *counterclockwise*])**

Adds points to the subpath such that the arc described by the circumference of the ellipse described by the arguments, starting at the given start angle and ending at the given end angle, going in the given direction (defaulting to clockwise), is added to the path, connected to the previous point by a straight line.

Throws an "IndexSizeError" DOMException if the given radius is negative.

***context*.rect[p743](*x*, *y*, *w*, *h*)**

***path*.rect[p743](*x*, *y*, *w*, *h*)**

Adds a new closed subpath to the path, representing the given rectangle.

***context*.roundRect[p744](*x*, *y*, *w*, *h*, *radii*)**

***path*.roundRect[p744](*x*, *y*, *w*, *h*, *radii*)**

Adds a new closed subpath to the path representing the given rounded rectangle. *radii* is either a list of radii or a single radius representing the corners of the rectangle in pixels. If a list is provided, the number and order of these radii function in the same way as the CSS 'border-radius' property. A single radius behaves the same way as a list with a single element.

If *w* and *h* are both greater than or equal to 0, or if both are smaller than 0, then the path is drawn clockwise. Otherwise, it is drawn counterclockwise.

When *w* is negative, the rounded rectangle is flipped horizontally, which means that the radius values that normally apply to the left corners are used on the right and vice versa. Similarly, when *h* is negative, the rounded rect is flipped vertically.

When a value *r* in *radii* is a number, the corresponding corner(s) are drawn as circular arcs of radius *r*.

When a value *r* in *radii* is an object with `{ x, y }` properties, the corresponding corner(s) are drawn as elliptical arcs whose *x* and *y* radii are equal to *r.x* and *r.y*, respectively.

When the sum of the *radii* of two corners of the same edge is greater than the length of the edge, all the *radii* of the rounded rectangle are scaled by a factor of length / (*r1* + *r2*). If multiple edges have this property, the scale factor of the edge with the smallest scale factor is used. This is consistent with CSS behavior.

Throws a RangeError if *radii* is a list whose size is not one, two, three, or four.

Throws a RangeError if a value in *radii* is a negative number, or is an `{ x, y }` object whose `x` or `y` properties are negative numbers.

The following methods allow authors to manipulate the paths[p739] of objects implementing the CanvasPath[p722] interface.

For objects implementing the CanvasDrawPath[p720] and CanvasTransform[p719] interfaces, the points passed to the methods, and the resulting lines added to current default path[p756] by these methods, must be transformed according to the current transformation matrix[p746] before being added to the path.

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[p739], the user agent must check to see if the path[p739] has its need new subpath[p739] 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()[p742] method had been called, and must then unset the path[p739]'s need new subpath[p739] 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.

Note

If the last subpath had more than one point in its list of points, then this is equivalent to adding a straight line connecting the last point back to the first point of the last subpath, thus "closing" the subpath.

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[p742] 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[p742] 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*). [BEZIER][p1579]
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[p742] 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*). [BEZIER][p1579]
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[p742] 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[p746] (so that it is in the same coordinate system as the points passed to the method).
5. If the point (*x0*, *y0*) is equal to the point (*x1*, *y1*), or if the point (*x1*, *y1*) is equal to the point (*x2*, *y2*), or if *radius* is zero, then add the point (*x1*, *y1*) to the subpath, and connect that point to the previous point (*x0*, *y0*) by a straight line.

6. Otherwise, if the points (*x0*, *y0*), (*x1*, *y1*), and (*x2*, *y2*) all lie on a single straight line, then add the point (*x1*, *y1*) to the subpath, and connect that point to the previous point (*x0*, *y0*) 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 (*x0*, *y0*) and ends at the point (*x1*, *y1*), and that has a different point tangent to the half-infinite line that ends at the point (*x1*, *y1*) and crosses the point (*x2*, *y2*). The points at which this circle touches these two lines are called the start and end tangent points respectively. Connect the point (*x0*, *y0*) 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[p743] with this, *x*, *y*, *radius*, *radius*, 0, *startAngle*, *endAngle*, and *counterclockwise*.

Note

This makes it equivalent to `ellipse()`[p743] *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[p743] with this, *x*, *y*, *radiusX*, *radiusY*, *rotation*, *startAngle*, *endAngle*, and *counterclockwise*.

The **determine the point on an ellipse steps**, given *ellipse*, and *angle*, are:

1. Let *eccentricCircle* be the circle that shares its origin with *ellipse*, with a radius equal to the semi-major axis of *ellipse*.
2. Let *outerPoint* be the point on *eccentricCircle*'s circumference at *angle* measured in radians clockwise from *ellipse*'s semi-major axis.
3. Let *chord* be the line perpendicular to *ellipse*'s major axis between this axis and *outerPoint*.
4. Return the point on *chord* that crosses *ellipse*'s circumference.

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 greater than or equal to 2π, or, if *counterclockwise* is *true* and *startAngle* − *endAngle* is greater than or equal to 2π, then the arc is the whole circumference of this ellipse, and both the start point and the end point are the result of running the determine the point on an ellipse steps[p743] given this ellipse and *startAngle*.

   Otherwise, the start point is the result of running the determine the point on an ellipse steps[p743] given this ellipse and *startAngle*, the end point is the result of running the determine the point on an ellipse steps[p743] given this ellipse and *endAngle*, 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 2π 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()`[p742] *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.

The **`roundRect(x, y, w, h, radii)`** method steps are:

✔ MDN

1. If any of *x*, *y*, *w*, or *h* are infinite or NaN, then return.
2. If *radii* is an `unrestricted double` or `DOMPointInit`, then set *radii* to « *radii* ».
3. If *radii* is not a list of size one, two, three, or four, then throw a `RangeError`.
4. Let *normalizedRadii* be an empty list.
5. For each *radius* of *radii*:
    1. If *radius* is a `DOMPointInit`:
        1. If *radius*["`x`"] or *radius*["`y`"] is infinite or NaN, then return.
        2. If *radius*["`x`"] or *radius*["`y`"] is negative, then throw a `RangeError`.
        3. Otherwise, append *radius* to *normalizedRadii*.
    2. If *radius* is a `unrestricted double`:
        1. If *radius* is infinite or NaN, then return.
        2. If *radius* is negative, then throw a `RangeError`.
        3. Otherwise, append «["`x`" → *radius*, "`y`" → *radius*]» to *normalizedRadii*.
6. Let *upperLeft*, *upperRight*, *lowerRight*, and *lowerLeft* be null.
7. If *normalizedRadii*'s size is 4, then set *upperLeft* to *normalizedRadii*[0], set *upperRight* to *normalizedRadii*[1], set *lowerRight* to *normalizedRadii*[2], and set *lowerLeft* to *normalizedRadii*[3].
8. If *normalizedRadii*'s size is 3, then set *upperLeft* to *normalizedRadii*[0], set *upperRight* and *lowerLeft* to *normalizedRadii*[1], and set *lowerRight* to *normalizedRadii*[2].
9. If *normalizedRadii*'s size is 2, then set *upperLeft* and *lowerRight* to *normalizedRadii*[0] and set *upperRight* and *lowerLeft* to *normalizedRadii*[1].
10. If *normalizedRadii*'s size is 1, then set *upperLeft*, *upperRight*, *lowerRight*, and *lowerLeft* to *normalizedRadii*[0].
11. Corner curves must not overlap. Scale all radii to prevent this:
    1. Let *top* be *upperLeft*["`x`"] + *upperRight*["`x`"].
    2. Let *right* be *upperRight*["`y`"] + *lowerRight*["`y`"].
    3. Let *bottom* be *lowerRight*["`x`"] + *lowerLeft*["`x`"].
    4. Let *left* be *upperLeft*["`y`"] + *lowerLeft*["`y`"].
    5. Let *scale* be the minimum value of the ratios *w* / *top*, *h* / *right*, *w* / *bottom*, *h* / *left*.
    6. If *scale* is less than 1, then set the `x` and `y` members of *upperLeft*, *upperRight*, *lowerLeft*, and *lowerRight* to their current values multiplied by *scale*.
12. Create a new subpath:
    1. Move to the point (*x* + *upperLeft*["`x`"], *y*).
    2. Draw a straight line to the point (*x* + *w* − *upperRight*["`x`"], *y*).
    3. Draw an arc to the point (*x* + *w*, *y* + *upperRight*["`y`"]).
    4. Draw a straight line to the point (*x* + *w*, *y* + *h* − *lowerRight*["`y`"]).

5. Draw an arc to the point ($x + w - lowerRight$["x"], $y + h$).
6. Draw a straight line to the point ($x + lowerLeft$["x"], $y + h$).
7. Draw an arc to the point (x, $y + h - lowerLeft$["y"]).
8. Draw a straight line to the point (x, $y + upperLeft$["y"]).
9. Draw an arc to the point ($x + upperLeft$["x"], y).

13. Mark the subpath as closed.
14. Create a new subpath with the point (x, y) as the only point in the subpath.

Note

This is designed to behave similarly to the CSS 'border-radius' property.

4.12.5.1.7 Path2D objects

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)

***path* = new Path2D()**

Creates a new empty Path2D object.

***path* = 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]

***path*.addPath(*path* [, *transform*])**

Adds to the path the path given by the argument.

The **Path2D(*path*)** 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(*path*, *transform*)** method, when invoked on a Path2D object *a*, must run these steps:

1. If the Path2D object *path* 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 *path*. Let *c* be this copy.
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.

4.12.5.1.8 Transformations §[p74 6]

Objects that implement the CanvasTransform[p719] interface have a **current transformation matrix**, as well as methods (described in this section) to manipulate it. When an object implementing the CanvasTransform[p719] interface is created, its transformation matrix must be initialized to the identity matrix.

The current transformation matrix[p746] is applied to coordinates when creating the current default path[p756], and when painting text, shapes, and Path2D[p723] objects, on objects implementing the CanvasTransform[p719] 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)

***context*.scale[p746](*x*, *y*)**

Changes the current transformation matrix[p746] to apply a scaling transformation with the given characteristics.

***context*.rotate[p746](*angle*)**

Changes the current transformation matrix[p746] to apply a rotation transformation with the given characteristics. The angle is in radians.

***context*.translate[p747](*x*, *y*)**

Changes the current transformation matrix[p746] to apply a translation transformation with the given characteristics.

***context*.transform[p747](*a*, *b*, *c*, *d*, *e*, *f*)**

Changes the current transformation matrix[p746] to apply the matrix given by the arguments as described below.

***matrix* = *context*.getTransform[p747]()**

Returns a copy of the current transformation matrix[p746], as a newly created DOMMatrix object.

***context*.setTransform[p747](*a*, *b*, *c*, *d*, *e*, *f*)**

Changes the current transformation matrix[p746] *to* the matrix given by the arguments as described below.

***context*.setTransform[p747](*transform*)**

Changes the current transformation matrix[p746] *to* the matrix represented by the passed DOMMatrix2DInit dictionary.

***context*.resetTransform[p747]()**

Changes the current transformation matrix[p746] 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[p746]. 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[p746]. 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[p746]. 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[p746] with the result of multiplying the current transformation matrix[p746] with the matrix described by:

a	*c*	*e*
b	*d*	*f*
0	0	1

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.*

The **getTransform()** method, when invoked, must return a newly created DOMMatrix representing a copy of the current transformation matrix[p746] matrix of the context.

Note

This returned object is not live, so updating it will not affect the current transformation matrix[p746], and updating the current transformation matrix[p746] 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[p746] to the matrix described by:

a	*c*	*e*
b	*d*	*f*
0	0	1

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[p746] to *matrix*.

The **resetTransform()** method, when invoked, must reset the current transformation matrix[p746] to the identity matrix.

Note

Given a matrix of the form created by the transform()[p747] *and* setTransform()[p747] *methods, i.e.,*

a	c	e
b	d	f
0	*0*	*1*

the resulting transformed coordinates after transform matrix multiplication will be

$x_{new} = a\ x + c\ y + e$
$y_{new} = b\ x + d\ y + f$

4.12.5.1.9 Image sources for 2D rendering contexts §[p748]

Some methods on the CanvasDrawImage[p721] and CanvasFillStrokeStyles[p720] interfaces take the union type CanvasImageSource[p718] as an argument.

This union type allows objects implementing any of the following interfaces to be used as image sources:

- HTMLOrSVGImageElement[p718] (img[p359] or SVG image elements)
- HTMLVideoElement[p421] (video[p420] elements)
- HTMLCanvasElement[p714] (canvas[p713] elements)
- OffscreenCanvas[p777]
- ImageBitmap[p1275]
- VideoFrame

Note

Although not formally specified as such, SVG image elements are expected to be implemented nearly identical to img[p359] elements. That is, SVG image elements share the fundamental concepts and features of img[p359] elements.

Note

The ImageBitmap[p1275] interface can be created from a number of other image-representing types, including ImageData[p1272].

To **check the usability of the *image* argument**, where *image* is a CanvasImageSource[p718] object, run these steps:

1. Switch on *image*:

   ↪ **HTMLOrSVGImageElement[p718]**

   If *image*'s current request[p377]'s state[p377] is broken[p377], then throw an "InvalidStateError" DOMException.

   If *image* is not fully decodable[p377], then return *bad*.

   If *image* has a natural width or natural height (or both) equal to zero, then return *bad*.

   ↪ **HTMLVideoElement[p421]**

   If *image*'s readyState[p452] attribute is either HAVE_NOTHING[p450] or HAVE_METADATA[p450], then return *bad*.

   ↪ **HTMLCanvasElement[p714]**

   ↪ **OffscreenCanvas[p777]**

   If *image* has either a horizontal dimension or a vertical dimension equal to zero, then throw an "InvalidStateError" DOMException.

   ↪ **ImageBitmap[p1275]**

   ↪ **VideoFrame**

   If *image*'s [[Detached]][p124] internal slot value is set to true, then throw an "InvalidStateError" DOMException.

2. Return *good*.

When a CanvasImageSource[p718] object represents an HTMLOrSVGImageElement[p718], the element's image must be used as the source image.

Specifically, when a CanvasImageSource[p718] object represents an animated image in an HTMLOrSVGImageElement[p718], 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[p719] APIs.

When a CanvasImageSource[p718] object represents an HTMLVideoElement[p421], then the frame at the current playback position[p448] when the method with the argument is invoked must be used as the source image when rendering the image for CanvasRenderingContext2D[p719] APIs, and the source image's dimensions must be the natural width[p424] and natural height[p424] of the media resource[p432] (i.e., after any aspect-ratio correction has been applied).

When a CanvasImageSource[p718] object represents an HTMLCanvasElement[p714], the element's bitmap must be used as the source image.

When a CanvasImageSource[p718] object represents an element that is being rendered[p1488] 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[p495] and height[p495] attributes on the source element have no effect on how the object is interpreted when rendering the image for CanvasRenderingContext2D[p719] APIs).

When a CanvasImageSource[p718] object represents an ImageBitmap[p1275], the object's bitmap image data must be used as the source image.

When a CanvasImageSource[p718] object represents an OffscreenCanvas[p777], the object's bitmap must be used as the source image.

When a CanvasImageSource[p718] object represents a VideoFrame, the object's pixel data must be used as the source image, and the source image's dimensions must be the object's [[display width]] and [[display height]].

An object *image* **is not origin-clean** if, switching on *image*'s type:

↪ **HTMLOrSVGImageElement**[p718]
image's current request[p377]'s image data[p377] is CORS-cross-origin[p102].

↪ **HTMLVideoElement**[p421]
image's media data[p431] is CORS-cross-origin[p102].

↪ **HTMLCanvasElement**[p714]
↪ **ImageBitmap**[p1275]
↪ **OffscreenCanvas**[p777]
image's bitmap's origin-clean[p715] flag is false.

4.12.5.1.10 Fill and stroke styles §[p749]

For web developers (non-normative)

***context*.fillStyle**[p750] **[= *value*]**

Returns the current style used for filling shapes.

Can be set, to change the fill style[p749].

The style can be either a string containing a CSS color, or a CanvasGradient[p722] or CanvasPattern[p722] object. Invalid values are ignored.

***context*.strokeStyle**[p750] **[= *value*]**

Returns the current style used for stroking shapes.

Can be set, to change the stroke style.[p749]

The style can be either a string containing a CSS color, or a CanvasGradient[p722] or CanvasPattern[p722] object. Invalid values are ignored.

Objects that implement the CanvasFillStrokeStyles[p720] interface have attributes and methods (defined in this section) that control how shapes are treated by the object.

Such objects have associated **fill style** and **stroke style** values, which are either CSS colors, CanvasPattern[p722]s, or CanvasGradient[p722]s. Initially, both must be the result of parsing the string "#000000".

When the value is a CSS color, it must not be affected by the transformation matrix when used to draw on bitmaps.

Note

When set to a CanvasPattern[p722] or CanvasGradient[p722] object, changes made to the object after the assignment do affect subsequent stroking or filling of shapes.

The **fillStyle** getter steps are:

1. If this's fill style[p749] is a CSS color, then return the serialization of that color with HTML-compatible serialization requested.
2. Return this's fill style[p749].

The fillStyle[p750] setter steps are:

1. If the given value is a string:
    1. Let *context* be this's canvas[p724] attribute's value, if that is an element; otherwise null.
    2. Let *parsedValue* be the result of parsing the given value with *context* if non-null.
    3. If *parsedValue* is failure, then return.
    4. Set this's fill style[p749] to *parsedValue*.
    5. Return.
2. If the given value is a CanvasPattern[p722] object that is marked as not origin-clean[p752], then set this's origin-clean[p715] flag to false.
3. Set this's fill style[p749] to the given value.

The **strokeStyle** getter steps are:

1. If this's stroke style[p749] is a CSS color, then return the serialization of that color with HTML-compatible serialization requested.
2. Return this's stroke style[p749].

The strokeStyle[p750] setter steps are:

1. If the given value is a string:
    1. Let *context* be this's canvas[p724] attribute's value, if that is an element; otherwise null.
    2. Let *parsedValue* be the result of parsing the given value with *context* if non-null.
    3. If *parsedValue* is failure, then return.
    4. Set this's stroke style[p749] to *parsedValue*.
    5. Return.
2. If the given value is a CanvasPattern[p722] object that is marked as not origin-clean[p752], then set this's origin-clean[p715] flag to false.
3. Set this's stroke style[p749] to the given value.

There are three types of gradients, linear gradients, radial gradients, and conic gradients, represented by objects implementing the opaque CanvasGradient[p722] 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, converted to the context's color space[p726]. Between each such stop, the colors and the alpha component must be linearly interpolated in the context's color space[p726] 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.

For web developers (non-normative)

***gradient*.addColorStop[p751](*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.

***gradient* = *context*.createConicGradient(*startAngle*, *x*, *y*)**

Returns a CanvasGradient object that represents a conic gradient that paints clockwise along the rotation around the center represented by the arguments.

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 $x_0 = x_1$ and $y_0 = y_1$ and $r_0 = r_1$, then the radial gradient must paint nothing. Return.
2. Let $x(\omega) = (x_1-x_0)\omega + x_0$.

   Let $y(\omega) = (y_1-y_0)\omega + y_0$.

   Let $r(\omega) = (r_1-r_0)\omega + r_0$.

   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 $r(\omega) > 0$, 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 $r(\omega)$ at position $(x(\omega), y(\omega))$, 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[p746] when rendering.

The **createConicGradient(*startAngle, x, y*)** method takes three arguments, the first argument, *startAngle*, represents the angle in radians at which the gradient begins, and the last two arguments, (*x*, *y*), represent the center of the gradient in CSS pixels. The method, when invoked, must return a conic CanvasGradient[p722] initialized with the specified center and angle.

It follows the same rendering rule as CSS 'conic-gradient' and it is equivalent to CSS 'conic-gradient(from *adjustedStartAngle*rad at *x*px *y*px, *angularColorStopList*)'. Here:

- *adjustedStartAngle* is given by *startAngle* + π/2;
- *angularColorStopList* is given by the color stops that have been added to the CanvasGradient[p722] using addColorStop()[p751], with the color stop offsets interpreted as percentages.

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[p722] interface.

For web developers (non-normative)

***pattern* = *context*.createPattern[p752](*image*, *repetition*)**

Returns a CanvasPattern[p722] 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.

***pattern*.setTransform[p752](*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[p748] *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[p722] object with the image *image* and the repetition behavior given by *repetition*.
7. If *image* is not origin-clean[p749], then mark *pattern* as **not origin-clean**.
8. Return *pattern*.

Modifying the *image* used when creating a CanvasPattern[p722] object after calling the createPattern()[p752] method must not affect the pattern(s) rendered by the CanvasPattern[p722] 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`[p767] 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`[p767] 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[p746].
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.

4.12.5.1.11 Drawing rectangles to the bitmap §[p753]

Objects that implement the `CanvasRect`[p720] 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[p746] 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[p756], and are subject to the clipping region[p757], and, with the exception of `clearRect()`[p753], also shadow effects[p767], global alpha[p766], and the current compositing and blending operator[p766].

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[p757].
3. Clear the pixels in *pixels* to a 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 this's fill style[p749].

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[p729] described below, using the `CanvasPathDrawingStyles`[p721] interface's line styles, and fill it with this's stroke style[p749].

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[p729] 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.

✔ MDN

4.12.5.1.12 Drawing text to the bitmap §[p754]

For web developers (non-normative)

***context*.fillText[p754](*text*, *x*, *y* [, *maxWidth*])**

***context*.strokeText[p754](*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.

***metrics* = *context*.measureText[p755](*text*)**

Returns a `TextMetrics`[p722] object with the metrics of the given text in the current font.

***metrics*.width[p755]**

***metrics*.actualBoundingBoxLeft[p755]**

***metrics*.actualBoundingBoxRight[p755]**

***metrics*.fontBoundingBoxAscent[p755]**

***metrics*.fontBoundingBoxDescent[p755]**

***metrics*.actualBoundingBoxAscent[p755]**

***metrics*.actualBoundingBoxDescent[p756]**

***metrics*.emHeightAscent[p756]**

***metrics*.emHeightDescent[p756]**

***metrics*.hangingBaseline[p756]**

***metrics*.alphabeticBaseline[p756]**

***metrics*.ideographicBaseline[p756]**

Returns the measurement described below.

Objects that implement the `CanvasText`[p721] interface provide the following methods for rendering text.

The **`fillText(text, x, y, maxWidth)`** and **`strokeText(text, x, y, maxWidth)`** methods 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[p733], textAlign[p734], and textBaseline[p734] 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[p737], passing it *text*, the object implementing the CanvasText[p721] 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[p746], with each CSS pixel in the coordinate space of *glyphs* mapped to one coordinate space unit.

   For fillText()[p754], this's fill style[p749] must be applied to the shapes and this's stroke style[p749] must be ignored. For strokeText()[p754], the reverse holds: this's stroke style[p749] must be applied to the result of tracing[p729] the shapes using the object implementing the CanvasText[p721] interface for the line styles, and this's fill style[p749] must be ignored.

   These shapes are painted without affecting the current path, and are subject to shadow effects[p767], global alpha[p766], the clipping region[p757], and the current compositing and blending operator[p766].

The **measureText(*text*)** method steps are to run the text preparation algorithm[p737], passing it *text* and the object implementing the CanvasText[p721] interface, and then using the returned inline box return a new TextMetrics[p722] object with members behaving as described in the following list: [CSS][p1580]

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[p734] 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[p755]) can be wider than the width of the inline box (width[p755]), 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[p734] 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[p734] 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[p755] 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[p734] 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[p734] 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[p755] 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[p734] value "top[p735]") 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`[p734] 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`[p734] attribute to the em-over baseline in the inline box, in CSS pixels; positive numbers indicating that the given baseline is below the em-over baseline (so this value will usually be positive). Zero if the given baseline is the em-over baseline; half the font size if the given baseline is halfway between the em-over baseline and the em-under baseline.

`emHeightDescent` attribute

The distance from the horizontal line indicated by the `textBaseline`[p734] attribute to the em-under baseline in the inline box, in CSS pixels; positive numbers indicating that the given baseline is above the em-under baseline. (Zero if the given baseline is the em-under baseline.)

`hangingBaseline` attribute

The distance from the horizontal line indicated by the `textBaseline`[p734] 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`[p734] 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`[p734] 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()`[p754] and `strokeText()`[p754] can spill out of the box given by the font size and the width returned by `measureText()`[p755] (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.

4.12.5.1.13 Drawing paths to the canvas §[p756]

Objects that implement the `CanvasDrawPath`[p720] interface have a **current default path**. There is only one current default path[p756], it is not part of the drawing state[p726]. The current default path[p756] is a path[p739], as described above.

For web developers (non-normative)

***context*.beginPath**[p757]**()**

Resets the current default path[p756].

***context*.fill**[p757]**([*fillRule*])**

***context*.fill**[p757]**(*path* [, *fillRule*])**

Fills the subpaths of the current default path[p756] or the given path with the current fill style, obeying the given fill rule.

***context*.stroke**[p757]**()**

***context*.stroke**[p757]**(*path*)**

Strokes the subpaths of the current default path[p756] or the given path with the current stroke style.

***context*.clip([*fillRule*])**

***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*])**

***context*.isPointInPath(*path*, *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 steps are to empty the list of subpaths in this's current default path so that it once again has zero subpaths.

Where the following method definitions use the term **intended path** for a Path2D-or-null *path*, it means *path* itself if it is a Path2D object, 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(*fillRule*)** method steps are to run the fill steps given this, null, and *fillRule*.

The **fill(*path*, *fillRule*)** method steps are to run the fill steps given this, *path*, and *fillRule*.

The **fill steps**, given a CanvasDrawPath *context*, a Path2D-or-null *path*, and a fill rule *fillRule*, are to fill all the subpaths of the intended path for *path*, using *context*'s fill style, and using the fill rule indicated by *fillRule*. Open subpaths must be implicitly closed when being filled (without affecting the actual subpaths).

The **stroke()** method steps are to run the stroke steps given this and null.

The **stroke(*path*)** method steps are to run the stroke steps given this and *path*.

The **stroke steps**, given a CanvasDrawPath *context* and a Path2D-or-null *path*, are to trace the intended path for *path*, using *context*'s line styles as set by its CanvasPathDrawingStyles mixin, and then fill the resulting path using *context*'s stroke style, using the nonzero winding rule.

Note

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.

Note

The stroke style is affected by the transformation during painting, even if the current default path is used.

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 the current compositing and blending operator. (The effect of transformations is described above and varies based on which path is being used.)

The **clip(*fillRule*)** method steps are to run the clip steps given this, null, and *fillRule*.

The **clip(*path*, *fillRule*)** method steps are to run the clip steps given this, *path*, and *fillRule*.

The **clip steps**, given a CanvasDrawPath *context*, a Path2D-or-null *path*, and a fill rule *fillRule*, are to create a new **clipping**

region by calculating the intersection of *context*'s current clipping region and the area described by the intended path[p757] for *path*, using the fill rule[p725] indicated by *fillRule*. 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, its current clipping region must be set to the largest infinite surface (i.e. by default, no clipping occurs).

The **`isPointInPath(x, y, fillRule)`** method steps are to return the result of the is point in path steps[p758] given this, null, *x*, *y*, and *fillRule*.

The **`isPointInPath(path, x, y, fillRule)`** method steps are to return the result of the is point in path steps[p758] given this, *path*, *x*, *y*, and *fillRule*.

The **is point in path steps**, given a CanvasDrawPath[p720] *context*, a Path2D[p723]-or-null *path*, two numbers *x* and *y*, and a fill rule[p725] *fillRule*, are:

1. If *x* or *y* are infinite or NaN, then return false.
2. If the point given by the *x* and *y* coordinates, when treated as coordinates in the canvas coordinate space unaffected by the current transformation, is inside the intended path[p757] for *path* as determined by the fill rule[p725] indicated by *fillRule*, then return true. 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.
3. Return false.

The **`isPointInStroke(x, y)`** method steps are to return the result of the is point in stroke steps[p758] given this, null, *x*, and *y*.

The **`isPointInStroke(path, x, y)`** method steps are to return the result of the is point in stroke steps[p758] given this, *path*, *x*, and *y*.

The **is point in stroke steps**, given a CanvasDrawPath[p720] *context*, a Path2D[p723]-or-null *path*, and two numbers *x* and *y*, are:

1. If *x* or *y* are infinite or NaN, then return false.
2. If the point given by the *x* and *y* coordinates, when treated as coordinates in the canvas coordinate space unaffected by the current transformation, is inside the path that results from tracing[p729] the intended path[p757] for *path*, using the nonzero winding rule[p725], and using *context*'s line styles as set by its CanvasPathDrawingStyles[p721] mixin, then return true. Points on the resulting path must be considered to be inside the path.
3. Return false.

¶[p758] Example

This canvas[p713] element has a couple of checkboxes. The path-related commands are highlighted:

```
<canvas height=400 width=750>
 <label><input type=checkbox id=showA> Show As</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.rect(x-5, y-5, 10, 10);
     context.stroke();
     if (element.checked) {
       context.fillStyle = 'black';
```

```
      context.fill();
    }
    context.fillText(element.labels[0].textContent, x+5, y);
  }
  context.beginPath();
  context.rect(x-7, y-7, 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);
  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>
```

4.12.5.1.14 Drawing focus rings §[p75 9]

For web developers (non-normative)

***context*.drawFocusIfNeeded[p760](*element*)**

If *element* is focused[p875], draws a focus ring around the current default path[p756], following the platform conventions for focus rings.

***context*.drawFocusIfNeeded(*path*, *element*)**

If *element* is focused, draws a focus ring around *path*, following the platform conventions for focus rings.

Objects that implement the CanvasUserInterface interface provide the following methods to draw focus rings.

The **drawFocusIfNeeded(*element*)** method steps are to draw focus if needed given this, *element*, and this's current default path.

The **drawFocusIfNeeded(*path*, *element*)** method steps are to draw focus if needed given this, *element*, and *path*.

To **draw focus if needed**, given an object implementing CanvasUserInterface *context*, an element *element*, and a path *path*:

1. If *element* is not focused or is not a descendant of *context*'s canvas element, then return.
2. Draw a focus ring of the appropriate style along *path*, following platform conventions.

   Note

   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 current compositing and blending operator, the fill style, the stroke style, or any of the members in the 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.

Note

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.

"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.

4.12.5.1.15 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)`

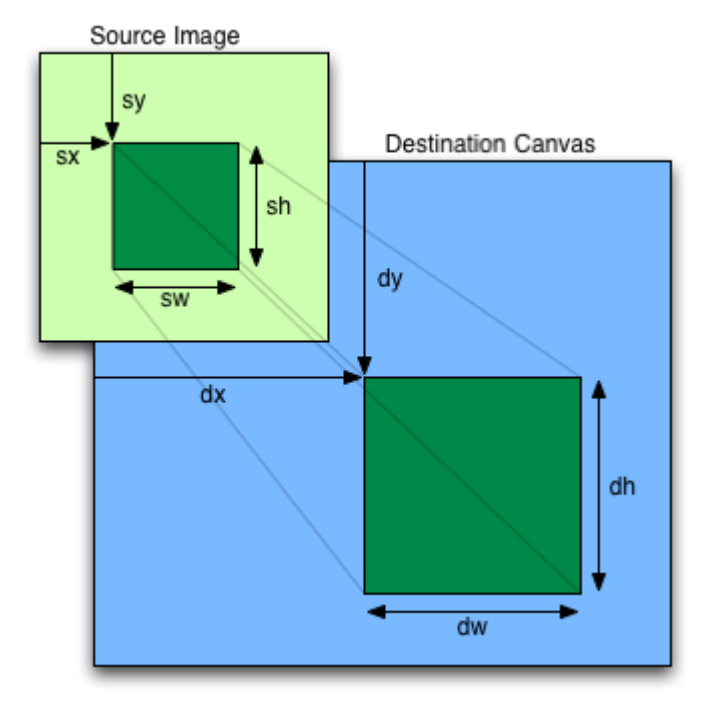
For web developers (non-normative)

***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()[p760] method is invoked, the user agent must run these steps:

1. If any of the arguments are infinite or NaN, then return.
2. Let *usability* be the result of checking the usability of *image*[p748].
3. If *usability* is *bad*, then return (without drawing anything).
4. Establish the source and destination rectangles as follows:

   If not specified, the *dw* and *dh* arguments must default to the values of *sw* and *sh*, interpreted such that one CSS pixel in the image is treated as one unit in the output bitmap[p725]'s coordinate space. If the *sx*, *sy*, *sw*, and *sh* arguments are omitted, then they must default to 0, 0, the image's natural width in image pixels, and the image's natural height in image pixels, respectively. If the image has no natural 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 natural properties being those of the *image* argument, and the default object size being the size of the output bitmap[p725]. [CSSIMAGES][p1581]

   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.

   Note

   When the destination rectangle is outside the destination image (the output bitmap[p725]), the pixels that land outside the output bitmap[p725] are discarded, as if the destination was an infinite canvas whose rendering was clipped to the dimensions of the output bitmap[p725].

5. If one of the *sw* or *sh* arguments is zero, then return. Nothing is painted.
6. Paint the region of the *image* argument specified by the source rectangle on the region of the rendering context's output bitmap[p725] specified by the destination rectangle, after applying the current transformation matrix[p746] 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 imageSmoothingEnabled[p767] 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 imageSmoothingQuality[p767] attribute to guide the choice of filtering algorithm when the imageSmoothingEnabled[p767] attribute is set to true. Otherwise, the image must be rendered using nearest-neighbor interpolation.

Note

This specification does not define the precise algorithm to use when scaling an image down, or when scaling an image up when the imageSmoothingEnabled[p767] attribute is set to true.

Note

When a canvas[p713] element is drawn onto itself, the drawing model[p769] requires the source to be copied before the image is drawn, so it is possible to copy parts of a canvas[p713] element onto overlapping parts of itself.

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.

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 canvas[p713] elements and drawing the scaled sprites from there.

Images are painted without affecting the current path, and are subject to shadow effects[p767], global alpha[p766], the clipping region[p757], and the current compositing and blending operator[p766].

7. If *image* is not origin-clean[p749], then set the CanvasRenderingContext2D[p719]'s origin-clean[p715] flag to false.

4.12.5.1.16 Pixel manipulation §[p762]

For web developers (non-normative)

***imageData* = *context*.createImageData[p763](*imageData*)**

Returns an ImageData[p1272] object with the same dimensions and color space as the argument. All the pixels in the returned object are transparent black.

***imageData* = *context*.createImageData[p763](*sw*, *sh* [, *settings*])**

Returns an ImageData[p1272] object with the given dimensions. The color space of the returned object is the color space[p726] of *context* unless overridden by *settings*. All the pixels in the returned object are transparent black.

Throws an "IndexSizeError" DOMException if either of the width or height arguments are zero.

***imageData* = *context*.getImageData[p763](*sx*, *sy*, *sw*, *sh* [, *settings*])**

Returns an ImageData[p1272] object containing the image data for the given rectangle of the bitmap. The color space of the returned object is the color space[p726] of *context* unless overridden by *settings*.

Throws an "IndexSizeError" DOMException if the either of the width or height arguments are zero.

***context*.putImageData[p763](*imageData*, *dx*, *dy* [, *dirtyX*, *dirtyY*, *dirtyWidth*, *dirtyHeight*])**

Paints the data from the given ImageData[p1272] object onto the bitmap. If a dirty rectangle is provided, only the pixels from that rectangle are painted.

The globalAlpha[p766] and globalCompositeOperation[p766] properties, as well as the shadow attributes[p767], 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[p1274] attribute value's [[ViewedArrayBuffer]] internal slot is detached.

Objects that implement the CanvasImageData[p721] interface provide the following methods for reading and writing pixel data to the bitmap.

The **createImageData(*sw*, *sh*, *settings*)** method steps are:

1. If one or both of *sw* and *sh* are zero, then throw an "IndexSizeError" DOMException.
2. Let *newImageData* be a new ImageData[p1272] object.
3. Initialize[p1274] *newImageData* given the absolute magnitude of *sw*, the absolute magnitude of *sh*, *settings*, and *defaultColorSpace*[p1274] set to this's color space[p726].
4. Initialize the image data of *newImageData* to transparent black.
5. Return *newImageData*.

The **createImageData(*imageData*)** method steps are:

1. Let *newImageData* be a new ImageData[p1272] object.
2. Let *settings* be the ImageDataSettings[p1272] object «["colorSpace[p1274]" → this's colorSpace[p1274], "pixelFormat[p1274]" → this's pixelFormat[p1274]]».
3. Initialize[p1274] *newImageData* given the value of *imageData*'s width[p1274] attribute, the value of *imageData*'s height[p1274] attribute, and *settings*.
4. Initialize the image data of *newImageData* to transparent black.
5. Return *newImageData*.

The **getImageData(*sx*, *sy*, *sw*, *sh*, *settings*)** method steps are:

1. If either the *sw* or *sh* arguments are zero, then throw an "IndexSizeError" DOMException.
2. If the CanvasRenderingContext2D[p719]'s origin-clean[p715] flag is set to false, then throw a "SecurityError" DOMException.
3. Let *imageData* be a new ImageData[p1272] object.
4. Initialize[p1274] *imageData* given *sw*, *sh*, *settings*, and *defaultColorSpace*[p1274] set to this's color space[p726].
5. Let the source rectangle be the rectangle whose corners are the four points (*sx*, *sy*), (*sx*+*sw*, *sy*), (*sx*+*sw*, *sy*+*sh*), (*sx*, *sy*+*sh*).
6. Set the pixel values of *imageData* to be the pixels of this's output bitmap[p725] in the area specified by the source rectangle in the bitmap's coordinate space units, converted from this's color space[p726] to *imageData*'s colorSpace[p1274] using 'relative-colorimetric' rendering intent.
7. Set the pixels values of *imageData* for areas of the source rectangle that are outside of the output bitmap[p725] to transparent black.
8. Return *imageData*.

The **putImageData(*imageData*, *dx*, *dy*)** method steps are to put pixels from an ImageData onto a bitmap[p763], given *imageData*, this's output bitmap[p725], *dx*, *dy*, 0, 0, *imageData*'s width[p1274], and *imageData*'s height[p1274].

The **putImageData(*imageData*, *dx*, *dy*, *dirtyX*, *dirtyY*, *dirtyWidth*, *dirtyHeight*)** method steps are to put pixels from an ImageData onto a bitmap[p763], given *imageData*, this's output bitmap[p725], *dx*, *dy*, *dirtyX*, *dirtyY*, *dirtyWidth*, and *dirtyHeight*.

To **put pixels from an ImageData onto a bitmap**, given an ImageData[p1272] *imageData*, an output bitmap[p725] *bitmap*, and numbers *dx*, *dy*, *dirtyX*, *dirtyY*, *dirtyWidth*, and *dirtyHeight*:

1. Let *buffer* be *imageData*'s data[p1274] 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 0.

   If *dirtyY* is negative, then let *dirtyHeight* be *dirtyHeight*+*dirtyY*, and let *dirtyY* be 0.

5. If *dirtyX*+*dirtyWidth* is greater than the `width`[p1274] attribute of the *imageData* argument, then let *dirtyWidth* be the value of that `width`[p1274] attribute, minus the value of *dirtyX*.

   If *dirtyY*+*dirtyHeight* is greater than the `height`[p1274] attribute of the *imageData* argument, then let *dirtyHeight* be the value of that `height`[p1274] 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*, set the pixel with coordinate (*dx*+*x*, *dy*+*y*) in *bitmap* to the color of the pixel at coordinate (*x*, *y*) in the *imageData* data structure's bitmap[p1273], converted from *imageData*'s `colorSpace`[p1274] to the color space[p726] of *bitmap* using 'relative-colorimetric' rendering intent.

Note

Due to the lossy nature of converting between color spaces and converting to and from premultiplied alpha[p784] color values, pixels that have just been set using `putImageData()`[p763], and are not completely opaque, might be returned to an equivalent `getImageData()`[p763] as different values.

The current path, transformation matrix[p746], shadow attributes[p767], global alpha[p766], the clipping region[p757], and current compositing and blending operator[p766] must not affect the methods described in this section.

Example

In the following example, the script generates an `ImageData`[p1272] object so that it can draw onto it.

```
// 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) { ... }
```

Example

Here is an example of using `getImageData()`[p763] and `putImageData()`[p763] to implement an edge detection filter.

```
<!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;
           outputData[i] = 127 + -inputData[i - w*4 - 4] -   inputData[i - w*4] - inputData[i -
w*4 + 4] +
                                -inputData[i - 4]       + 8*inputData[i]       - inputData[i + 4]
+
                                -inputData[i + w*4 - 4] -   inputData[i + w*4] - inputData[i +
w*4 + 4];
         }
         outputData[(y*w + x)*4 + 3] = 255; // alpha
       }
     }

     // put the image data back after manipulation
     context.putImageData(output, 0, 0);
   }
  </script>
 </head>
 <body onload="init()">
  <canvas></canvas>
 </body>
</html>
```

Example

Here is an example of color space conversion applied when drawing a solid color and reading the result back using and getImageData()[p763].

```
<!DOCTYPE HTML>
<html lang="en">
<title>Color space image data demo</title>

<canvas></canvas>

<script>
const canvas = document.querySelector('canvas');
```

```
const context = canvas.getContext('2d', {colorSpace:'display-p3'});

// Draw a red rectangle. Note that the hex color notation
// specifies sRGB colors.
context.fillStyle = "#FF0000";
context.fillRect(0, 0, 64, 64);

// Get the image data.
const pixels = context.getImageData(0, 0, 1, 1);

// This will print 'display-p3', reflecting the default behavior
// of returning image data in the canvas's color space.
console.log(pixels.colorSpace);

// This will print the values 234, 51, and 35, reflecting the
// red fill color, converted to 'display-p3'.
console.log(pixels.data[0]);
console.log(pixels.data[1]);
console.log(pixels.data[2]);
</script>
```

4.12.5.1.17 Compositing §

For web developers (non-normative)

***context*.globalAlpha** [= *value*]

Returns the current global alpha value applied to rendering operations.

Can be set, to change the global alpha value. Values outside of the range 0.0 .. 1.0 are ignored.

***context*.globalCompositeOperation** [= *value*]

Returns the current compositing and blending operator, from the values defined in *Compositing and Blending*. [COMPOSITE]

Can be set, to change the current compositing and blending operator. Unknown values are ignored.

Objects that implement the CanvasCompositing interface have a global alpha value and a current compositing and blending operator value that both affect all the drawing operations on this object.

The **global alpha** value gives an alpha value that is applied to shapes and images before they are composited onto the output bitmap. The value ranges from 0.0 (fully transparent) to 1.0 (no additional transparency). It must initially have the value 1.0.

The **globalAlpha** getter steps are to return this's global alpha.

The globalAlpha setter steps are:

1. If the given value is either infinite, NaN, or not in the range 0.0 to 1.0, then return.
2. Otherwise, set this's global alpha to the given value.

The **current compositing and blending operator** value controls how shapes and images are drawn onto the output bitmap, once they have had the global alpha and the current transformation matrix applied. Initially, it must be set to "source-over".

The **globalCompositeOperation** getter steps are to return this's current compositing and blending operator.

The globalCompositeOperation setter steps are:

1. If the given value is not identical to any of the values that the <blend-mode> or the <composite-mode> properties are defined to take, then return. [COMPOSITE]
2. Otherwise, set this's current compositing and blending operator to the given value.

4.12.5.1.18 Image smoothing §p767

For web developers (non-normative)

***context*.imageSmoothingEnabled**p767 **[= *value*]**

Returns whether pattern fills and the drawImage()p760 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).

***context*.imageSmoothingQuality**p767 **[= *value*]**

Returns the current image-smoothing-quality preference.

Can be set, to change the preferred quality of image smoothing. The possible values are "lowp725", "mediump725" and "highp725". Unknown values are ignored.

Objects that implement the CanvasImageSmoothingp720 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 CanvasImageSmoothingp720 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 CanvasImageSmoothingp720 interface is created, the attribute must be set to "lowp725".

4.12.5.1.19 Shadows §p767

All drawing operations on an object which implements the CanvasShadowStylesp720 interface are affected by the four global shadow attributes.

For web developers (non-normative)

***context*.shadowColor**p767 **[= *value*]**

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**p767 **[= *value*]**

***context*.shadowOffsetY**p767 **[= *value*]**

Returns the current shadow offset.

Can be set, to change the shadow offset. Values that are not finite numbers are ignored.

***context*.shadowBlur**p768 **[= *value*]**

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.

Objects which implement the CanvasShadowStylesp720 interface have an associated **shadow color**, which is a CSS color. Initially, it must be transparent black.

The **shadowColor** getter steps are to return the serialization of this's shadow colorp767 with HTML-compatible serialization requested.

The shadowColorp767 setter steps are:

1. Let *context* be this's canvasp724 attribute's value, if that is an element; otherwise null.
2. Let *parsedValue* be the result of parsing the given value with *context* if non-null.
3. If *parsedValue* is failure, then return.
4. Set this's shadow colorp767 to *parsedValue*.

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[p768] 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 shadow color[p767] is nonzero and either the shadowBlur[p768] is nonzero, or the shadowOffsetX[p767] is nonzero, or the shadowOffsetY[p767] is nonzero.

When shadows are drawn[p768], 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 component of *A* to *B*, offset by shadowOffsetX[p767] in the positive *x* direction, and shadowOffsetY[p767] in the positive *y* direction.
4. If shadowBlur[p768] is greater than 0:
    1. Let σ be half the value of shadowBlur[p768].
    2. Perform a 2D Gaussian Blur on *B*, using σ as the standard deviation.

    User agents may limit values of σ to an implementation-specific maximum value to avoid exceeding hardware limitations during the Gaussian blur operation.
5. Set the red, green, and blue components of every pixel in *B* to the red, green, and blue components (respectively) of the shadow color[p767].
6. Multiply the alpha component of every pixel in *B* by the alpha component of the shadow color[p767].
7. The shadow is in the bitmap *B*, and is rendered as part of the drawing model[p769] described below.

If the current compositing and blending operator[p766] is "copy", then shadows effectively won't render (since the shape will overwrite the shadow).

4.12.5.1.20 Filters §[p768]

All drawing operations on an object which implements the CanvasFilters[p720] interface are affected by the global **filter** attribute.

For web developers (non-normative)

***context*.filter[p768] [= *value*]**

Returns the current filter.

Can be set, to change the filter. Values can either be the string "none" or a string parseable as a <filter-value-list>. Other values are ignored.

Such objects have an associated **current filter**, which is a string. Initially the current filter[p768] is set to the string "none". Whenever the value of the current filter[p768] is the string "none" filters will be disabled for the context.

The filter[p768] getter steps are to return this's current filter[p768].

The filter[p768] setter steps are:

1. If the given value is "none", then set this's current filter[p768] to "none" and return.
2. Let *parsedValue* be the result of parsing the given values as a <filter-value-list>. If any property-independent style sheet syntax like 'inherit' or 'initial' is present, then this parsing must return failure.

3. If *parsedValue* is failure, then return.

4. Set this's current filter[p768] to the given value.

Note

Though `context.filter[p768] = "none"` *will disable filters for the context,* `context.filter[p768] = ""`, `context.filter[p768] = null`, *and* `context.filter[p768] = undefined` *are all treated as unparseable inputs and the value of the current filter[p768] is left unchanged.*

Coordinates used in the value of the current filter[p768] 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[p746]. The current transformation matrix affects only the input to the filter. Filters are applied in the output bitmap[p725]'s coordinate space.

When the value of the current filter[p768] is a string parsable as a <filter-value-list> which 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[p732] at the time that the attribute is set. If the computed values are undefined for a particular case (e.g. because the font style source object[p732] is not an element or is not being rendered[p1488]), then the relative keywords must be interpreted relative to the default value of the `font`[p733] attribute. The 'larger' and 'smaller' keywords are not supported.

If the value of the current filter[p768] is a string parseable as a <filter-value-list> with a reference 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 current filter[p768] is a string parseable as a <filter-value-list> with a reference 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.

4.12.5.1.21 Working with externally-defined SVG filters §[p769]

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.

4.12.5.1.22 Drawing model §[p769]

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. Convert the image *A* to the context's color space[p726].

   Note

   There do not exist any inputs to a 2D context for which the color space is undefined. The color space for CSS colors is defined in CSS Color. *The color space for images that specify no color profile information is 'srgb', as specified in the Color Spaces of Untagged Colors section of* CSS Color. *[CSSCOLOR][p1580]*

3. Multiply the alpha component of every pixel in *A* by `global_alpha`[p766].

4. When the current filter[p768] 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 current filter[p768], creating image *B*. If the current filter[p768] is a string parseable as a <filter-value-list>, then draw using the current filter[p768] in the same manner as SVG.

   Otherwise, let *B* be an alias for *A*.

5. When shadows are drawn[p768], render the shadow from image *B*, using the current shadow styles, creating image *C*.

6. When shadows are drawn[p768], composite *C* within the clipping region[p757] over the current output bitmap[p725] using the current

compositing and blending operator[p766].

7. Composite *B* within the clipping region[p757] over the current output bitmap[p725] using the current compositing and blending operator[p766].

When compositing onto the output bitmap[p725], pixels that would fall outside of the output bitmap[p725] must be discarded.

4.12.5.1.23 Best practices §[p77 0]

When a canvas is interactive, authors should include focusable[p876] elements in the element's fallback content corresponding to each focusable[p876] part of the canvas, as in the example above[p758].

When rendering focus rings, to ensure that focus rings have the appearance of native focus rings, authors should use the `drawFocusIfNeeded()`[p760] method, passing it the element for which a ring is being drawn. This method only draws the focus ring if the element is focused[p875], so that it can simply be called whenever drawing the element, without checking whether the element is focused or not first.

Authors should avoid implementing text editing controls using the `canvas`[p713] 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`[p539] element, the `textarea`[p606] element, or the `contenteditable`[p892] attribute.

4.12.5.1.24 Examples §[p77 0]

This section is non-normative.

Example

Here is an example of a script that uses canvas to draw pretty glowing lines.

```
<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);

 function blank() {
   context.fillStyle = 'rgba(0,0,0,0.1)';
   context.fillRect(0, 0, context.canvas.width, context.canvas.height);
 }
 setInterval(blank, 40);

</script>
```

Example

The 2D rendering context for canvas[p713] is often used for sprite-based games. The following example demonstrates this:

Here is the source for this example:

```
<!DOCTYPE HTML>
<html lang="en">
<meta charset="utf-8">
<title>Blue Robot Demo</title>
<style>
  html { overflow: hidden; min-height: 200px; min-width: 380px; }
  body { height: 200px; position: relative; margin: 8px; }
  .buttons { position: absolute; bottom: 0px; left: 0px; margin: 4px; }
</style>
<canvas width="380" height="200"></canvas>
<script>
 var Landscape = function (context, width, height) {
   this.offset = 0;
   this.width = width;
   this.advance = function (dx) {
     this.offset += dx;
   };
   this.horizon = height * 0.7;
   // This creates the sky gradient (from a darker blue to white at the bottom)
   this.sky = context.createLinearGradient(0, 0, 0, this.horizon);
   this.sky.addColorStop(0.0, 'rgb(55,121,179)');
   this.sky.addColorStop(0.7, 'rgb(121,194,245)');
   this.sky.addColorStop(1.0, 'rgb(164,200,214)');
   // this creates the grass gradient (from a darker green to a lighter green)
   this.earth = context.createLinearGradient(0, this.horizon, 0, height);
   this.earth.addColorStop(0.0, 'rgb(81,140,20)');
   this.earth.addColorStop(1.0, 'rgb(123,177,57)');
   this.paintBackground = function (context, width, height) {
     // first, paint the sky and grass rectangles
     context.fillStyle = this.sky;
     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.lineStyle = '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.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.lineStyle = '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';
  };
  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';
        }
      }
    }
```

```
    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.paint(context, canvas.width/2, landscape.horizon*1.1);
   landscape.paintForeground(context, canvas.width, canvas.height);
   requestAnimationFrame(paint);
 }
 paint();
 // but tick every 100ms, so that we don't slow down when we don't paint
 setInterval(function () {
   var dx = blueRobot.tick();
   landscape.advance(dx);
 }, 100);
</script>
<p class="buttons">
 <input type=button value="Walk" onclick="blueRobot.walk()">
 <input type=button value="Stop" onclick="blueRobot.stop()">
<footer>
 <small> Blue Robot Player Sprite by <a href="https://johncolburn.deviantart.com/">JohnColburn</a>.
 Licensed under the terms of the Creative Commons Attribution Share-Alike 3.0 Unported
license.</small>
 <small> This work is itself licensed under a <a rel="license" href="https://creativecommons.org/
licenses/by-sa/3.0/">Creative
 Commons Attribution-ShareAlike 3.0 Unported License</a>.</small>
</footer>
```

4.12.5.2 The ImageBitmap[p1275] rendering context §[p77]

4.12.5.2.1 Introduction §[p77]

ImageBitmapRenderingContext[p775] is a performance-oriented interface that provides a low overhead method for displaying the contents of ImageBitmap[p1275] objects. It uses transfer semantics to reduce overall memory consumption. It also streamlines performance by avoiding intermediate compositing, unlike the drawImage()[p760] method of CanvasRenderingContext2D[p719].

Using an img[p359] 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[p359] 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[p775].

Example

Using ImageBitmapRenderingContext[p775], here is how to transcode an image to the JPEG format in a memory- and CPU-efficient way:

```
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[p775] interface §[p775]

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)

***context* = *canvas*.getContext[p716]('bitmaprenderer' [, { [alpha[p776]: false] }])**

Returns an ImageBitmapRenderingContext[p775] object that is permanently bound to a particular canvas[p713] element.

If the alpha[p776] setting is provided and set to false, then the canvas is forced to always be opaque.

***context*.canvas[p775]**

Returns the canvas[p713] element that the context is bound to.

***context*.transferFromImageBitmap[p776](imageBitmap)**

Transfers the underlying bitmap data[p1276] from *imageBitmap* to *context*, and the bitmap becomes the contents of the canvas[p713] element to which *context* is bound.

***context*.transferFromImageBitmap[p776](null)**

Replaces contents of the canvas[p713] element to which *context* is bound with a transparent black bitmap whose size corresponds to the width[p715] and height[p715] content attributes of the canvas[p713] element.

The **canvas** attribute must return the value it was initialized to when the object was created.

An ImageBitmapRenderingContext[p775] object has an **output bitmap**, which is a reference to bitmap data[p1276].

An ImageBitmapRenderingContext[p775] object has a **bitmap mode**, which can be set to **valid** or **blank**. A value of valid[p775] indicates that the context's output bitmap[p775] refers to bitmap data[p1276] that was acquired via transferFromImageBitmap()[p776]. A value blank[p775] indicates that the context's output bitmap[p775] is a default transparent bitmap.

An ImageBitmapRenderingContext[p775] object also has an **alpha** flag, which can be set to true or false. When an ImageBitmapRenderingContext[p775] object has its alpha[p775] flag set to false, the contents of the canvas[p713] element to which the context is bound are obtained by compositing the context's output bitmap[p775] onto an opaque black bitmap of the same size using the source-over compositing operator. If the alpha[p775] flag is set to true, then the output bitmap[p775] is used as the contents of the canvas[p713] element to which the context is bound. [COMPOSITE][p1579]

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`[p775] object and an optional argument *bitmap* that refers to bitmap data[p1276], it must run these steps:

1. If a *bitmap* argument was not provided:
    1. Set *context*'s bitmap mode[p775] to blank[p775].
    2. Let *canvas* be the `canvas`[p713] element to which *context* is bound.
    3. Set *context*'s output bitmap[p775] to be transparent black with a natural width equal to the numeric value[p715] of *canvas*'s `width`[p715] attribute and a natural height equal to the numeric value[p715] of *canvas*'s `height`[p715] attribute, those values being interpreted in CSS pixels.
    4. Set the output bitmap[p775]'s origin-clean[p715] flag to true.
2. If a *bitmap* argument was provided:
    1. Set *context*'s bitmap mode[p775] to valid[p775].
    2. Set *context*'s output bitmap[p775] to refer to the same underlying bitmap data as *bitmap*, without making a copy.

    Note

    The origin-clean[p715] flag of bitmap *is included in the bitmap data to be referenced by* context's *output bitmap[p775].*

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`[p775]. (This can throw an exception.)
2. Let *context* be a new `ImageBitmapRenderingContext`[p775] object.
3. Initialize *context*'s `canvas`[p724] attribute to point to *target*.
4. Set *context*'s output bitmap[p775] to the same bitmap as *target*'s bitmap (so that they are shared).
5. Run the steps to set an `ImageBitmapRenderingContext`'s output bitmap[p776] with *context*.
6. Initialize *context*'s alpha[p775] flag to true.
7. Process each of the members of *settings* as follows:

    `alpha`

    If false, then set *context*'s alpha[p775] flag to false.

8. Return *context*.

The **`transferFromImageBitmap(bitmap)`** method, when invoked, must run these steps:

1. Let *bitmapContext* be the `ImageBitmapRenderingContext`[p775] object on which the `transferFromImageBitmap()`[p776] method was called.
2. If *bitmap* is null, then run the steps to set an ImageBitmapRenderingContext's output bitmap[p776], with *bitmapContext* as the *context* argument and no *bitmap* argument, then return.
3. If the value of *bitmap*'s [[Detached]][p124] internal slot is set to true, then throw an `"InvalidStateError"` `DOMException`.
4. Run the steps to set an `ImageBitmapRenderingContext`'s output bitmap[p776], with the *context* argument equal to *bitmapContext*, and the *bitmap* argument referring to *bitmap*'s underlying bitmap data[p1276].

5. Set the value of *bitmap*'s [[Detached]]p124 internal slot to true.

6. Unset *bitmap*'s bitmap datap1276.

4.12.5.3 The OffscreenCanvasp777 interface §p777

IDL

```
typedef (OffscreenCanvasRenderingContext2D or ImageBitmapRenderingContext or WebGLRenderingContext or
WebGL2RenderingContext or GPUCanvasContext) OffscreenRenderingContext;

dictionary ImageEncodeOptions {
  DOMString type = "image/png";
  unrestricted double quality;
};

enum OffscreenRenderingContextId { "2d", "bitmaprenderer", "webgl", "webgl2", "webgpu" };

[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 = {});

  attribute EventHandler oncontextlost;
  attribute EventHandler oncontextrestored;
};
```

Note

OffscreenCanvasp777 is an EventTarget, so both OffscreenCanvasRenderingContext2Dp781 and WebGL can fire events at it. OffscreenCanvasRenderingContext2Dp781 can fire contextlostp1575 and contextrestoredp1575, and WebGL can fire webglcontextlost *and* webglcontextrestored*. [WEBGL]p1588*

OffscreenCanvasp777 objects are used to create rendering contexts, much like an HTMLCanvasElementp714, but with no connection to the DOM. This makes it possible to use canvas rendering contexts in workersp1306.

An OffscreenCanvasp777 object may hold a weak reference to a **placeholder canvas element**, which is typically in the DOM, whose embedded content is provided by the OffscreenCanvasp777 object. The bitmap of the OffscreenCanvasp777 object is pushed to the placeholder canvas elementp777 as part of the OffscreenCanvasp777's relevant agentp1141's event loopp1193's update the renderingp1198 steps.

For web developers (non-normative)

***offscreenCanvas* = new OffscreenCanvasp778(*width*, *height*)**

Returns a new OffscreenCanvasp777 object that is not linked to a placeholder canvas elementp777, and whose bitmap's size is determined by the *width* and *height* arguments.

***context* = *offscreenCanvas*.getContextp778(*contextId* [, *options*])**

Returns an object that exposes an API for drawing on the OffscreenCanvasp777 object. *contextId* specifies the desired API: "2dp779", "bitmaprendererp779", "webglp779", "webgl2p779", or "webgpup779". *options* is handled by that API.

This specification defines the "2dp716" context below, which is similar but distinct from the "2dp779" context that is created from a canvasp713 element. The WebGL specifications define the "webglp779" and "webgl2p779" contexts. *WebGPU* defines the "webgpup779" context. [WEBGL]p1588 [WEBGPU]p1588

Returns null if the canvas has already been initialized with another context type (e.g., trying to get a "2dp779" context after getting a "webglp779" context).

An `OffscreenCanvas`[p777] object has an internal **bitmap** that is initialized when the object is created. The width and height of the bitmap[p778] are equal to the values of the `width`[p779] and `height`[p779] attributes of the `OffscreenCanvas`[p777] object. Initially, all the bitmap's pixels are transparent black.

An `OffscreenCanvas`[p777] object has an internal **inherited language** and **inherited direction** set when the `OffscreenCanvas`[p777] is created.

An `OffscreenCanvas`[p777] 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`[p777] object also has a **context mode**, which is initially **none** but can be changed to either **2d**, **bitmaprenderer**, **webgl**, **webgl2**, **webgpu**, or **detached** by algorithms defined in this specification.

The **`new OffscreenCanvas(width, height)`** constructor steps are:

1. Initialize the bitmap[p778] of this to a rectangular array of transparent black pixels of the dimensions specified by *width* and *height*.
2. Initialize the `width`[p779] of this to *width*.
3. Initialize the `height`[p779] of this to *height*.
4. Set this's inherited language[p778] to explicitly unknown[p166].
5. Set this's inherited direction[p778] to "`ltr`".
6. Let *global* be the relevant global object[p1151] of this.
7. If *global* is a `Window`[p965] object:
    1. Let *element* be the document element of *global*'s associated `Document`[p966].
    2. If *element* is not null:
        1. Set the inherited language[p778] of this to *element*'s language[p166].
        2. Set the inherited direction[p778] of this to *element*'s directionality[p168].

`OffscreenCanvas`[p777] objects are transferable[p123]. Their transfer steps[p123], given *value* and *dataHolder*, are as follows:

1. If *value*'s context mode[p778] is not equal to none[p778], then throw an "`InvalidStateError`" `DOMException`.
2. Set *value*'s context mode[p778] to detached[p778].
3. Let *width* and *height* be the dimensions of *value*'s bitmap[p778].
4. Let *language* and *direction* be the values of *value*'s inherited language[p778] and inherited direction[p778].
5. Unset *value*'s bitmap[p778].
6. Set *dataHolder*.[[Width]] to *width* and *dataHolder*.[[Height]] to *height*.
7. Set *dataHolder*.[[Language]] to *language* and *dataHolder*.[[Direction]] to *direction*.
8. Set *dataHolder*.[[PlaceholderCanvas]] to be a weak reference to *value*'s placeholder canvas element[p777], if *value* has one, or null if it does not.

Their transfer-receiving steps[p123], given *dataHolder* and *value*, are:

1. Initialize *value*'s bitmap[p778] to a rectangular array of transparent black pixels with width given by *dataHolder*.[[Width]] and height given by *dataHolder*.[[Height]].
2. Set *value*'s inherited language[p778] to *dataHolder*.[[Language]] and its inherited direction[p778] to *dataHolder*.[[Direction]].
3. If *dataHolder*.[[PlaceholderCanvas]] is not null, set *value*'s placeholder canvas element[p777] to *dataHolder*.[[PlaceholderCanvas]] (while maintaining the weak reference semantics).

The **`getContext(contextId, options)`** method of an `OffscreenCanvas`[p777] 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 `OffscreenCanvas`[p777] object's context mode[p778] and whose row header matches *contextId*:

	none[p778]	**2d**[p778]	**bitmaprenderer**[p778]	**webgl**[p778] **or** **webgl2**[p778]	**webgpu**[p778]	**detached**[p778]
"2d"	1. Let *context* be the result of running the offscreen 2D context creation algorithm[p781] given this and *options*. 2. Set this's context mode[p778] to 2d[p778]. 3. Return *context*.	Return the same object as was returned the last time the method was invoked with this same first argument.	Return null.	Return null.	Return null.	Throw an "InvalidStateError" DOMException.
"bitmaprenderer"	1. Let *context* be the result of running the ImageBitmapRenderingContext creation algorithm[p776] given this and *options*. 2. Set this's context mode[p778] to bitmaprenderer[p778]. 3. Return *context*.	Return null.	Return the same object as was returned the last time the method was invoked with this same first argument.	Return null.	Return null.	Throw an "InvalidStateError" DOMException.
"webgl" or "webgl2"	1. Let *context* be the result of following the instructions given in the WebGL specifications' *Context Creation* sections. [WEBGL][p1588] 2. If *context* is null, then return null; otherwise set this's context mode[p778] to webgl[p778] or webgl2[p778]. 3. Return *context*.	Return null.	Return null.	Return the same value as was returned the last time the method was invoked with this same first argument.	Return null.	Throw an "InvalidStateError" DOMException.
"webgpu"	1. Let *context* be the result of following the instructions given in *WebGPU*'s Canvas Rendering section. [WEBGPU][p1588] 2. If *context* is null, then return null; otherwise set this's context mode[p778] to webgpu[p778]. 3. Return *context*.	Return null.	Return null.	Return null.	Return the same value as was returned the last time the method was invoked with this same first argument.	Throw an "InvalidStateError" DOMException.

For web developers (non-normative)

***offscreenCanvas*.width**[p779] **[= *value*]**

***offscreenCanvas*.height**[p779] **[= *value*]**

These attributes return the dimensions of the `OffscreenCanvas`[p777] object's bitmap[p778].

They can be set, to replace the bitmap[p778] with a new, transparent black bitmap of the specified dimensions (effectively resizing it).

If either the **width** or **height** attributes of an `OffscreenCanvas`[p777] object are set (to a new value or to the same value as before) and

the OffscreenCanvas[p777] object's context mode[p778] is 2d[p778], then reset the rendering context to its default state[p727] and resize the OffscreenCanvas[p777] object's bitmap[p778] to the new values of the width[p779] and height[p779] attributes.

The resizing behavior for "webgl[p779]" and "webgl2[p779]" contexts is defined in the WebGL specifications. [WEBGL][p1588]

The resizing behavior for "webgpu[p779]" context is defined in *WebGPU*. [WEBGPU][p1588]

Note

If an OffscreenCanvas[p777] object whose dimensions were changed has a placeholder canvas element[p777], then the placeholder canvas element[p777]'s natural size will only be updated during the OffscreenCanvas[p777]'s relevant agent[p1141]'s event loop[p1193]'s update the rendering[p1198] steps.

For web developers (non-normative)

***promise* = *offscreenCanvas*.convertToBlob[p780](*options*)**

Returns a promise that will fulfill with a new Blob object representing a file containing the image in the OffscreenCanvas[p777] object.

The argument, if provided, is a dictionary that controls the encoding options of the image file to be created. The type[p780] field specifies the file format and has a default value of "image/png[p1577]"; that type is also used if the requested type isn't supported. If the image format supports variable quality (such as "image/jpeg[p1577]"), then the quality[p780] field is a number in the range 0.0 to 1.0 inclusive indicating the desired quality level for the resulting image.

***canvas*.transferToImageBitmap[p780]()**

Returns a newly created ImageBitmap[p1275] object with the image in the OffscreenCanvas[p777] object. The image in the OffscreenCanvas[p777] object is replaced with a new blank image.

The **convertToBlob(*options*)** method steps are:

1. If the value of this's [[Detached]][p124] internal slot is true, then return a promise rejected with an "InvalidStateError" DOMException.
2. If this's context mode[p778] is 2d[p778] and the rendering context's output bitmap[p725]'s origin-clean[p715] flag is set to false, then return a promise rejected with a "SecurityError" DOMException.
3. If this's bitmap[p778] 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's bitmap[p778].
5. Let *result* be a new promise object.
6. Let *global* be this's relevant global object[p1151].
7. Run these steps in parallel[p44]:
    1. Let *file* be a serialization of *bitmap* as a file[p782], with *options*'s **type** and **quality** if present.
    2. Queue a global task[p1195] on the canvas blob serialization task source[p718] given *global* 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 *global*'s relevant realm[p1151], representing *file*. [FILEAPI][p1582]
8. Return *result*.

The **transferToImageBitmap()** method, when invoked, must run the following steps:

1. If the value of this OffscreenCanvas[p777] object's [[Detached]][p124] internal slot is set to true, then throw an "InvalidStateError" DOMException.
2. If this OffscreenCanvas[p777] object's context mode[p778] is set to none[p778], then throw an "InvalidStateError" DOMException.
3. Let *image* be a newly created ImageBitmap[p1275] object that references the same underlying bitmap data as this OffscreenCanvas[p777] object's bitmap[p778].

4. Set this OffscreenCanvas[p777] object's bitmap[p778] to reference a newly created bitmap of the same dimensions and color space as the previous bitmap, and with its pixels initialized to transparent black, or opaque black if the rendering context's alpha[p725] is false.

   Note

   This means that if the rendering context of this OffscreenCanvas[p777] is a WebGLRenderingContext, the value of preserveDrawingBuffer will have no effect. [WEBGL][p1588]

5. Return *image*.

The following are the event handlers[p1206] (and their corresponding event handler event types[p1209]) that must be supported, as event handler IDL attributes[p1207], by all objects implementing the OffscreenCanvas[p777] interface:

Event handler[p1206]	Event handler event type[p1209]
oncontextlost	contextlost[p1575]
oncontextrestored	contextrestored[p1575]

4.12.5.3.1 The offscreen 2D rendering context §[p781]

IDL

```
[Exposed=(Window,Worker)]
interface OffscreenCanvasRenderingContext2D {
  readonly attribute OffscreenCanvas canvas;
};

OffscreenCanvasRenderingContext2D includes CanvasSettings;
OffscreenCanvasRenderingContext2D includes CanvasState;
OffscreenCanvasRenderingContext2D includes CanvasTransform;
OffscreenCanvasRenderingContext2D includes CanvasCompositing;
OffscreenCanvasRenderingContext2D includes CanvasImageSmoothing;
OffscreenCanvasRenderingContext2D includes CanvasFillStrokeStyles;
OffscreenCanvasRenderingContext2D includes CanvasShadowStyles;
OffscreenCanvasRenderingContext2D includes CanvasFilters;
OffscreenCanvasRenderingContext2D includes CanvasRect;
OffscreenCanvasRenderingContext2D includes CanvasDrawPath;
OffscreenCanvasRenderingContext2D includes CanvasText;
OffscreenCanvasRenderingContext2D includes CanvasDrawImage;
OffscreenCanvasRenderingContext2D includes CanvasImageData;
OffscreenCanvasRenderingContext2D includes CanvasPathDrawingStyles;
OffscreenCanvasRenderingContext2D includes CanvasTextDrawingStyles;
OffscreenCanvasRenderingContext2D includes CanvasPath;
```

The OffscreenCanvasRenderingContext2D[p781] object is a rendering context for drawing to the bitmap[p778] of an OffscreenCanvas[p777] object. It is similar to the CanvasRenderingContext2D[p719] object, with the following differences:

- there is no support for user interface[p721] features;
- its canvas[p782] attribute refers to an OffscreenCanvas[p777] object rather than a canvas[p713] element;

An OffscreenCanvasRenderingContext2D[p781] object has an **associated OffscreenCanvas object**, which is the OffscreenCanvas[p777] object from which the OffscreenCanvasRenderingContext2D[p781] object was created.

For web developers (non-normative)

***offscreenCanvas* = *offscreenCanvasRenderingContext2D*.canvas[p782]**

Returns the associated OffscreenCanvas object[p781].

The **offscreen 2D context creation algorithm**, which is passed a *target* (an OffscreenCanvas[p777] 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 *arg* to the dictionary type CanvasRenderingContext2DSettings[p718]. (This can throw an exception.)
3. Let *context* be a new OffscreenCanvasRenderingContext2D[p781] object.
4. Set *context*'s associated OffscreenCanvas object[p781] to *target*.
5. Run the canvas settings output bitmap initialization algorithm[p726], given *context* and *settings*.
6. Set *context*'s output bitmap[p725] to a newly created bitmap with the dimensions specified by the width[p779] and height[p779] attributes of *target*, and set *target*'s bitmap to the same bitmap (so that they are shared).
7. If *context*'s alpha[p725] flag is set to true, initialize all the pixels of *context*'s output bitmap[p725] to transparent black. Otherwise, initialize the pixels to opaque black.
8. Return *context*.

Note

Implementations are encouraged to short-circuit the graphics update steps of the window event loop[p1193] for the purposes of updating the contents of a placeholder canvas element[p777] to the display. This could mean, for example, that the bitmap contents are copied directly to a graphics buffer that is mapped to the physical display location of the placeholder canvas element[p777]. This or similar short-circuiting approaches can significantly reduce display latency, especially in cases where the OffscreenCanvas[p777] is updated from a worker event loop[p1193] and the window event loop[p1193] of the placeholder canvas element[p777] is busy. However, such shortcuts cannot have any script-observable side-effects. This means that the committed bitmap still needs to be sent to the placeholder canvas element[p777], in case the element is used as a CanvasImageSource[p718], as an ImageBitmapSource[p1275], or in case toDataURL()[p717] or toBlob()[p718] are called on it.

The **canvas** attribute, on getting, must return this OffscreenCanvasRenderingContext2D[p781]'s associated OffscreenCanvas object[p781].

4.12.5.4 Color spaces and color space conversion §[p78]₂

```
IDL
enum PredefinedColorSpace { "srgb", "srgb-linear", "display-p3", "display-p3-linear" };
```

The PredefinedColorSpace[p782] enumeration is used to specify the color space[p726] of the canvas's backing store.

The "**srgb**" value indicates the 'srgb' color space.

The "**srgb-linear**" value indicates the 'srgb-linear' color space.

The "**display-p3**" value indicates the 'display-p3' color space.

The "**display-p3-linear**" value indicates the 'display-p3-linear' color space.

Color space conversion must be applied to the canvas's backing store when rendering the canvas to the output device.

Note

The algorithm for converting between color spaces can be found in the Converting Colors section of CSS Color*. This color space conversion is identical to the color space conversion that would be applied to an img[p359] element with a color profile that specifies the same color space[p726] as the canvas's backing store. [CSSCOLOR][p1580]*

4.12.5.5 Serializing bitmaps to a file §[p78]₂

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][p1585]

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[p1577]" would mean to generate a PNG image, the value "image/jpeg[p1577]" would mean to generate a JPEG image, and the value "image/svg+xml[p1577]" 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[p1577]"). 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][p1585]

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 component, the serialized image must be the bitmap image composited onto an opaque black background using the source-over compositing operator.

For image types that support color profiles, the serialized image must include a color profile indicating the color space of the underlying bitmap. For image types that do not support color profiles, the serialized image must be converted to the 'srgb' color space using 'relative-colorimetric' rendering intent.

Note

Thus, in the 2D context, calling the drawImage()[p760] *method to render the output of the* toDataURL()[p717] *or* toBlob()[p718] *method to the canvas, given the appropriate dimensions, has no visible effect beyond, at most, clipping colors of the canvas to a more narrow gamut.*

For image types that support multiple bit depths, the serialized image must use the bit depth that best preserves content of the underlying bitmap.

Example

For example, when serializing a 2D context that has color type[p726] of float16[p724] to *type* "image/png[p1577]", the resulting image would have 16 bits per sample. This serialization will still lose significant detail (all values less than 0.5/65535 would be clamped to 0, and all values greater than 1 would be clamped to 1).

If *type* is an image format that supports variable quality (such as "image/jpeg[p1577]"), *quality* is given, and *type* is not "image/png[p1577]", then, if *quality* is a Number 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[p713] elements §[p783]

This section is non-normative.

Information leakage can occur if scripts from one origin[p939] can access information (e.g. read pixels) from images from another origin (one that isn't the same[p940]).

To mitigate this, bitmaps used with canvas[p713] elements, OffscreenCanvas[p777] objects, and ImageBitmap[p1275] objects are defined to have a flag indicating whether they are origin-clean[p715]. All bitmaps start with their origin-clean[p715] set to true. The flag is set to false when cross-origin images are used.

The toDataURL()[p717], toBlob()[p718], and getImageData()[p763] methods check the flag and will throw a "SecurityError" DOMException rather than leak cross-origin data.

The value of the origin-clean[p715] flag is propagated from a source's bitmap to a new ImageBitmap[p1275] object by createImageBitmap()[p1276]. Conversely, a destination canvas[p713] element's bitmap will have its origin-clean[p715] flags set to false by drawImage[p760] if the source image is an ImageBitmap[p1275] object whose bitmap has its origin-clean[p715] flag set to false.

The flag can be reset in certain situations; for example, when changing the value of the width[p715] or the height[p715] content attribute of the canvas[p713] element to which a CanvasRenderingContext2D[p719] is bound, the bitmap is cleared and its origin-clean[p715] flag is reset.

When using an ImageBitmapRenderingContext[p775], the value of the origin-clean[p715] flag is propagated from ImageBitmap[p1275] objects when they are transferred to the canvas[p713] via transferFromImageBitmap()[p776].

4.12.5.7 Premultiplied alpha and the 2D rendering context §[p784]

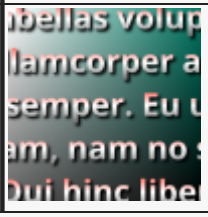
Premultiplied alpha refers to one way of representing transparency in an image, the other being non-premultiplied alpha.

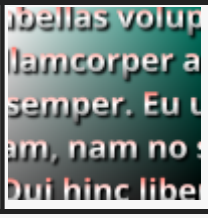
Under non-premultiplied alpha, the red, green, and blue components of a pixel represent that pixel's color, and its alpha component represents that pixel's opacity.

Under premultiplied alpha, however, the red, green, and blue components of a pixel represent the amounts of color that the pixel adds to the image, and its alpha component represents the amount that the pixel obscures whatever is behind it.

Example

For instance, assuming the color components range from 0 (off) to 255 (full intensity), these example colors are represented in the following ways:

CSS color representation	Premultiplied representation	Non-premultiplied representation	Description of color	Image of color blended above other content
rgba(255, 127, 0, 1)	255, 127, 0, 255	255, 127, 0, 255	Completely-opaque orange	
rgba(255, 255, 0, 0.5)	127, 127, 0, 127	255, 255, 0, 127	Halfway-opaque yellow	
Unrepresentable	255, 127, 0, 127	Unrepresentable	Additive halfway-opaque orange	
Unrepresentable	255, 127, 0, 0	Unrepresentable	Additive fully-transparent orange	
rgba(255, 127, 0, 0)	0, 0, 0, 0	255, 127, 0, 0	Fully-transparent ("invisible") orange	

CSS color representation	Premultiplied representation	Non-premultiplied representation	Description of color	Image of color blended above other content
rgba(0, 127, 255, 0)	0, 0, 0, 0	255, 127, 0, 0	Fully-transparent ("invisible") turquoise	

Converting a color value from a non-premultiplied representation to a premultiplied one involves multiplying the color's red, green, and blue components by its alpha component (remapping the range of the alpha component such that "fully transparent" is 0, and "fully opaque" is 1).

Converting a color value from a premultiplied representation to a non-premultiplied one involves the inverse: dividing the color's red, green, and blue components by its alpha component.

As certain colors can only be represented under premultiplied alpha (for instance, additive colors), and others can only be represented under non-premultiplied alpha (for instance, "invisible" colors which hold certain red, green, and blue values even with no opacity); and division and multiplication using finite precision entails a loss of accuracy, converting between premultiplied and non-premultiplied alpha is a lossy operation on colors that are not fully opaque.

A `CanvasRenderingContext2D`[p719]'s output bitmap[p725] and an `OffscreenCanvasRenderingContext2D`[p781]'s output bitmap[p725] must use premultiplied alpha to represent transparent colors.

Note

It is important for canvas bitmaps to represent colors using premultiplied alpha because it affects the range of representable colors. While additive colors cannot currently be drawn onto canvases directly because CSS colors are non-premultiplied and cannot represent them, it is still possible to, for instance, draw additive colors onto a WebGL canvas and then draw that WebGL canvas onto a 2D canvas via `drawImage()`[p760].

4.13 Custom elements §[p785]

4.13.1 Introduction §[p785]

This section is non-normative.

Custom elements[p796] 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[p800] 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 §[p785]

This section is non-normative.

For the purposes of illustrating how to create an autonomous custom element[p796], 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:

```
<flag-icon country="nl"></flag-icon>
```

To do this, we first declare a class for the custom element, extending HTMLElement[p149]:

```
class FlagIcon extends HTMLElement {
  constructor() {
    super();
    this._countryCode = null;
  }

  static observedAttributes = ["country"];

  attributeChangedCallback(name, oldValue, newValue) {
    // name will always be "country" due to observedAttributes
    this._countryCode = newValue;
    this._updateRendering();
  }
  connectedCallback() {
    this._updateRendering();
  }

  get country() {
    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:

```
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:

```
const flagIcon = document.createElement("flag-icon")
flagIcon.country = "jp"
document.body.appendChild(flagIcon)
```

Finally, we can also use the custom element constructor[p796] itself. That is, the above code is equivalent to:

```
const flagIcon = new FlagIcon()
flagIcon.country = "jp"
document.body.appendChild(flagIcon)
```

4.13.1.2 Creating a form-associated custom element §[p78 6]

This section is non-normative.

Adding a static `formAssociated` property, with a true value, makes an autonomous custom element[p796] a form-associated custom element[p797]. The `ElementInternals`[p809] interface helps you to implement functions and properties common to form control elements.

```
class MyCheckbox extends HTMLElement {
  static formAssociated = true;
  static observedAttributes = ['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.hasAttribute('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);
  }

  _onClick(event) {
    this.checked = !this.checked;
  }
}
customElements.define('my-checkbox', MyCheckbox);
```

You can use the custom element `my-checkbox` like a built-in form-associated element. For example, putting it in `form`[p532] or `label`[p536] associates the `my-checkbox` element with them, and submitting the `form`[p532] will send data provided by `my-checkbox` implementation.

```
<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 §[p787]

This section is non-normative.

By using the appropriate properties of `ElementInternals`[p809], 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:

```
class MyCheckbox extends HTMLElement {
  static formAssociated = true;
  static observedAttributes = ['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.hasAttribute('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[p70] and aria-*[p70] attributes:

```
<!-- This markup is non-conforming -->
<input type="checkbox" checked role="button" aria-checked="false">
```

```
<!-- This markup is probably not what the custom element author intended -->
<my-checkbox role="button" checked aria-checked="false">
```

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 §[p78]

This section is non-normative.

Customized built-in elements[p796] are a distinct kind of custom element[p796], which are defined slightly differently and used very differently compared to autonomous custom elements[p796]. 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[p796]. Instead, customized built-in elements[p796] 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[p796] require a distinct syntax from autonomous custom elements[p796] 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[p796] 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[p796] 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[p585] instead of HTMLElement[p149]:

```
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:

```
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:

```
<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():

```
const plasticButton = document.createElement("button", { is: "plastic-button" });
plasticButton.textContent = "Click me!";
```

And as before, the constructor will also work:

```
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:

```
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[p878] 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:

```
class TacoButton extends HTMLElement {
  static observedAttributes = ["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[p878] attributes of its own volition, and its choice of `tabindex="0"` focusability behavior may not match the button[p585] 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[p878] 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[p796], as shown in the previous section, would automatically inherit the semantics and behavior of the button[p585] 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[p796] will be easier to develop, maintain, and consume.

4.13.1.6 Upgrading elements after their creation §[p791]

This section is non-normative.

Because element definition[p800] can occur at any time, a non-custom element could be created, and then later become a custom element[p796] after an appropriate definition[p798] is registered. We call this process "upgrading" the element, from a normal element into a custom element.

Upgrades[p803] enable scenarios where it may be preferable for custom element definitions[p798] 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:

```
<!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[p685] element marked with the async[p687] 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[p309]. 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[p803] 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:

```
<!DOCTYPE html>
<html lang="en">
<title>Upgrade edge-cases example</title>

<example-element></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:
  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>
```

4.13.1.7 Scoped custom element registries §[p79]

To allow multiple libraries to co-exist without explicit coordination, CustomElementRegistry[p799] can be used in a scoped fashion as well.

```
const scoped = new CustomElementRegistry();
scoped.define("example-element", ExampleElement);

const element = document.createElement("example-element", { customElementRegistry: scoped });
```

A node with an associated scoped CustomElementRegistry[p799] will use that registry for all its operations, such as when invoking setHTMLUnsafe()[p1227].

4.13.1.8 Exposing custom element states §[p79]

Built-in elements provided by user agents have certain states that can change over time depending on user interaction and other factors, and are exposed to web authors through pseudo-classes. For example, some form controls have the "invalid" state, which is exposed through the :invalid[p823] pseudo-class.

Like built-in elements, custom elements[p796] can have various states to be in too, and custom element[p796] authors want to expose these states in a similar fashion as the built-in elements.

This is done via the :state()[p824] pseudo-class. A custom element author can use the states[p813] property of ElementInternals[p809] to add and remove such custom states, which are then exposed as arguments to the :state()[p824] pseudo-class.

Example

The following shows how :state()[p824] can be used to style a custom checkbox element. Assume that LabeledCheckbox doesn't expose its "checked" state via a content attribute.

```
<script>
class LabeledCheckbox extends HTMLElement {
  constructor() {
    super();
    this._internals = this.attachInternals();
    this.addEventListener('click', this._onClick.bind(this));

    const shadowRoot = this.attachShadow({mode: 'closed'});
    shadowRoot.innerHTML =
      `<style>
       :host::before {
         content: '[ ]';
         white-space: pre;
         font-family: monospace;
       }
       :host(:state(checked))::before { content: '[x]' }
       </style>
       <slot>Label</slot>`;
  }

  get checked() { return this._internals.states.has('checked'); }

  set checked(flag) {
    if (flag)
      this._internals.states.add('checked');
    else
      this._internals.states.delete('checked');
  }

  _onClick(event) {
    this.checked = !this.checked;
  }
}

customElements.define('labeled-checkbox', LabeledCheckbox);
</script>

<style>
labeled-checkbox { border: dashed red; }
labeled-checkbox:state(checked) { border: solid; }
</style>

<labeled-checkbox>You need to check this</labeled-checkbox>
```

Example

Custom pseudo-classes can even target shadow parts. An extension of the above example shows this:

```
<script>
class QuestionBox extends HTMLElement {
  constructor() {
    super();
    const shadowRoot = this.attachShadow({mode: 'closed'});
    shadowRoot.innerHTML =
      `<div><slot>Question</slot></div>
       <labeled-checkbox part='checkbox'>Yes</labeled-checkbox>`;
  }
}
```

```
customElements.define('question-box', QuestionBox);
</script>

<style>
question-box::part(checkbox) { color: red; }
question-box::part(checkbox):state(checked) { color: green; }
</style>

<question-box>Continue?</question-box>
```

4.13.2 Requirements for custom element constructors and reactions §[p794]

When authoring custom element constructors[p796], 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()`[p1223] or `document.open()`[p1221] methods.
- The element's attributes and children must not be inspected, as in the non-upgrade[p803] 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[p1194], as a microtask checkpoint[p1200] can occur immediately after construction.

When authoring custom element reactions[p806], 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:

```
class CParent extends HTMLElement {
  connectedCallback() {
    this.firstChild.remove();
  }
}
customElements.define("c-parent", CParent);

class CChild extends HTMLElement {
  connectedCallback() {
    console.log("CChild connectedCallback: isConnected =", this.isConnected);
  }
}
customElements.define("c-child", CChild);
```

```
const parent = new CParent(),
      child = new CChild();
parent.append(child);
document.body.append(parent);

// Logs:
// CChild connectedCallback: isConnected = false
```

4.13.2.1 Preserving custom element state when moved §[p795]

This section is non-normative.

When manipulating the DOM tree, an element can be moved in the tree while connected. This applies to custom elements as well. By default, the "disconnectedCallback" and "connectedCallback" would be called on the element, one after the other. This is done to maintain compatibility with existing custom elements that predate the moveBefore() method. This means that by default, custom elements reset their state as if they were removed and re-inserted. In the example above[p790], the impact would be that the observer would be disconnected and re-connected, and the tab index would be reset.

To opt in to a state-preserving behavior while moving, the author can implement a "connectedMoveCallback". The existence of this callback, even if empty, would supersede the default behavior of calling "disconnectedCallback" and "connectedCallback". "connectedMoveCallback" can also be an appropriate place to execute logic that depends on the element's ancestors. For example:

```
class TacoButton extends HTMLElement {
  static observedAttributes = ["disabled"];

  constructor() {
    super();
    this._internals = this.attachInternals();
    this._internals.role = "button";

    this._observer = new MutationObserver(() => {
      this._internals.ariaLabel = this.textContent;
    });
  }

  _notifyMain() {
    if (this.parentElement.tagName === "MAIN") {
      // Execute logic that depends on ancestors.
    }
  }

  connectedCallback() {
    this.setAttribute("tabindex", "0");

    this._observer.observe(this, {
      childList: true,
      characterData: true,
      subtree: true
    });

    this._notifyMain();
  }

  disconnectedCallback() {
    this._observer.disconnect();
  }

  // Implementing this function would avoid resetting the tab index or re-registering the
```

```
  // mutation observer when this element is moved inside the DOM without being disconnected.
  connectedMoveCallback() {
    // The parent can change during a state-preserving move.
    this._notifyMain();
  }
}
```

4.13.3 Core concepts §[p796]

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**.

MDN

Two distinct types of custom elements[p796] can be defined:

- An **autonomous custom element**, which is defined with no `extends`[p799] option. These types of custom elements have a local name equal to their defined name[p798].
- A **customized built-in element**, which is defined with an `extends`[p799] option. These types of custom elements have a local name equal to the value passed in their `extends`[p799] option, and their defined name[p798] is used as the value of the **`is`** attribute, which therefore must be a valid custom element name[p797].

After a custom element[p796] is created, changing the value of the `is`[p796] attribute does not change the element's behavior, as it is saved on the element as its `is` value.

Autonomous custom elements[p796] have the following element definition:

Categories[p154]:
Flow content[p157].
Phrasing content[p157].
Palpable content[p158].
For form-associated custom elements[p797]: Listed[p532], labelable[p532], submittable[p532], and resettable[p532] form-associated element[p532].

Contexts in which this element can be used[p154]:
Where phrasing content[p157] is expected.

Content model[p154]:
Transparent[p159].

Content attributes[p154]:
Global attributes[p162], except the `is`[p796] attribute
`form`[p627], for form-associated custom elements[p797] — Associates the element with a `form`[p532] element
`disabled`[p630], for form-associated custom elements[p797] — Whether the form control is disabled
`readonly`[p797], for form-associated custom elements[p797] — Affects `willValidate`[p654], plus any behavior added by the custom element author
`name`[p628], for form-associated custom elements[p797] — Name of the element to use for form submission[p657] and in the `form.elements`[p534] API
Any other attribute that has no namespace (see prose).

Accessibility considerations[p154]:
For form-associated custom elements[p797]: for authors; for implementers.
Otherwise: for authors; for implementers.

Sanitization[p154]:
Uncategorized[p1249].

DOM interface[p154]:
Supplied by the element's author (inherits from `HTMLElement`[p149])

An autonomous custom element[p796] does not have any special meaning: it represents[p148] its children. A customized built-in element[p796] 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[p796], so long as the attribute name is a valid attribute local name and contains no ASCII upper alphas. The exception is the `is`[p796] attribute, which must not be specified on an autonomous custom element[p796] (and which will have no effect if it is).

Customized built-in elements[p796] follow the normal requirements for attributes, based on the elements they extend. To add custom attribute-based behavior, use `data-*`[p171] attributes.

An autonomous custom element[p796] is called a **form-associated custom element** if the element is associated with a custom element definition[p798] whose form-associated[p798] field is set to true.

The `name`[p628] attribute represents the form-associated custom element[p797]'s name. The `disabled`[p630] attribute is used to make the form-associated custom element[p797] non-interactive and to prevent its submission value[p811] from being submitted. The `form`[p627] attribute is used to explicitly associate the form-associated custom element[p797] with its form owner[p626].

The **`readonly`** attribute of form-associated custom elements[p797] specifies that the element is barred from constraint validation[p651]. 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 readonly[p570] attribute on built-in form controls.

Constraint validation: If the `readonly`[p797] attribute is specified on a form-associated custom element[p797], the element is barred from constraint validation[p651].

The reset algorithm[p666] for form-associated custom elements[p797] is to enqueue a custom element callback reaction[p807] with the element, callback name "`formResetCallback`", and « ».

A string *name* is a **valid custom element name** if all of the following are true:

- *name* is a valid element local name;

  Note

  This ensures the custom element can be created with `createElement()`.

- *name*'s 0th code point is an ASCII lower alpha;

  Note

  This ensures the HTML parser will treat the name as a tag name instead of as text.

- *name* does not contain any ASCII upper alphas;

  Note

  This ensures the user agent can always treat HTML elements ASCII-case-insensitively.

- *name* contains a U+002D (-); and

  Note

  This is used for namespacing and to ensure forward compatibility (since no elements will be added to HTML, SVG, or MathML with hyphen-containing local names going forward).

- *name* is not one of the following:

  - `"annotation-xml"`
  - `"color-profile"`
  - `"font-face"`
  - `"font-face-src"`
  - `"font-face-uri"`
  - `"font-face-format"`
  - `"font-face-name"`
  - `"missing-glyph"`

  Note

  The list of names above is the summary of all hyphen-containing element names from the applicable specifications[p77], namely SVG 2 *and* MathML. *[SVG][p1586] [MATHML][p1584]*

Apart from these restrictions, a large variety of names is allowed, to give maximum flexibility for use cases like `<math-α>` or `<emotion-😍>`.

A **custom element definition** describes a custom element[p796] and consists of:

A *name*
: A valid custom element name[p797]

A *local name*
: A local name

A *constructor*
: A Web IDL `CustomElementConstructor`[p799] callback function type value wrapping the custom element constructor[p796]

A list of *observed attributes*
: A `sequence<DOMString>`

A collection of *lifecycle callbacks*
: A map, whose keys are the strings "`connectedCallback`", "`disconnectedCallback`", "`connectedMoveCallback`", "`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[p803] algorithm and the HTML element constructors[p151]. 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[p798] as form-associated custom elements[p797].

A *disable internals* **boolean**
: Controls `attachInternals()`[p809].

A *disable shadow* **boolean**
: Controls `attachShadow()`.

To **look up a custom element definition**, given null or a `CustomElementRegistry`[p799] object *registry*, string-or-null *namespace*, string *localName*, and string-or-null *is*, perform the following steps. They will return either a custom element definition[p798] or null:

1. If *registry* is null, then return null.
2. If *namespace* is not the HTML namespace, then return null.
3. If *registry*'s custom element definition set[p799] contains an item with name[p798] and local name[p798] both equal to *localName*, then return that item.
4. If *registry*'s custom element definition set[p799] contains an item with name[p798] equal to *is* and local name[p798] equal to *localName*, then return that item.
5. Return null.

4.13.4 The `CustomElementRegistry`[p799] interface §[p798]

Each similar-origin window agent[p1140] has an associated **active custom element constructor map**, which is a map of constructors to `CustomElementRegistry`[p799] objects.

The `Window`[p965] **`customElements`** getter steps are:

1. Assert: this's associated `Document`[p966]'s custom element registry is a `CustomElementRegistry`[p799] object.

Note

A `Window`[p965]'s associated `Document`[p966] is always created with a new `CustomElementRegistry`[p799] object.

2. Return this's associated Document[p966]'s custom element registry.

IDL

```
[Exposed=Window]
interface CustomElementRegistry {
  constructor();

  [CEReactions] undefined define(DOMString name, CustomElementConstructor constructor, optional
ElementDefinitionOptions options = {});
  (CustomElementConstructor or undefined) get(DOMString name);
  DOMString? getName(CustomElementConstructor constructor);
  Promise<CustomElementConstructor> whenDefined(DOMString name);
  [CEReactions] undefined upgrade(Node root);
  [CEReactions] undefined initialize(Node root);
};

callback CustomElementConstructor = HTMLElement ();

dictionary ElementDefinitionOptions {
  DOMString extends;
};
```

Every CustomElementRegistry[p799] has an **is scoped**, a boolean, initially false.

Every CustomElementRegistry[p799] has a **scoped document set**, a set of Document[p135] objects, initially « ».

Every CustomElementRegistry[p799] has a **custom element definition set**, a set of custom element definitions[p798], initially « ». Lookup of items in this set uses their name[p798], local name[p798], or constructor[p798].

Every CustomElementRegistry[p799] also has an **element definition is running** boolean which is used to prevent reentrant invocations of element definition[p800]. It is initially false.

Every CustomElementRegistry[p799] also has a **when-defined promise map**, a map of valid custom element names[p797] to promises. It is used to implement the whenDefined()[p802] method.

To **look up a custom element registry**, given a Node object *node*:

1. If *node* is an Element object, then return *node*'s custom element registry.
2. If *node* is a ShadowRoot object, then return *node*'s custom element registry.
3. If *node* is a Document[p135] object, then return *node*'s custom element registry.
4. Return null.

For web developers (non-normative)

***registry* = *window*.customElements[p798]**

Returns the global's associated Document[p135]'s CustomElementRegistry[p799] object.

***registry* = new CustomElementRegistry[p800]()**

Constructs a new CustomElementRegistry[p799] object, for scoped usage.

***registry*.define[p800](*name*, *constructor*)**

Defines a new custom element[p796], mapping the given name to the given constructor as an autonomous custom element[p796].

***registry*.define[p800](*name*, *constructor*, { extends: *baseLocalName* })**

Defines a new custom element[p796], mapping the given name to the given constructor as a customized built-in element[p796] for the element type[p46] identified by the supplied *baseLocalName*. A "NotSupportedError" DOMException will be thrown upon trying to extend a custom element[p796] or an unknown element, or when *registry* is not a global CustomElementRegistry[p799] object.

***registry*.get[p802](*name*)**

Retrieves the custom element constructor[p796] defined for the given name[p798]. Returns undefined if there is no custom element definition[p798] with the given name[p798].

***registry*.getName(*constructor*)**

Retrieves the given name for a custom element defined for the given constructor. Returns null if there is no custom element definition with the given constructor.

***registry*.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.

***registry*.upgrade(*root*)**

Tries to upgrade all shadow-including inclusive descendant elements of *root*, even if they are not connected.

***registry*.initialize(*root*)**

Each inclusive descendant of *root* with a null registry will have it updated to this CustomElementRegistry object. A "NotSupportedError" DOMException will be thrown if this CustomElementRegistry object is not for scoped usage and either *root* is a Document node or *root*'s node document's custom element registry is not this CustomElementRegistry object.

The **new CustomElementRegistry()** constructor steps are to set this's is scoped to true.

Element definition is a process of adding a custom element definition to the CustomElementRegistry. This is accomplished by the define() method.

The **define(*name*, *constructor*, *options*)** method steps are:

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's custom element definition set contains an item with name *name*, then throw a "NotSupportedError" DOMException.
4. If this's custom element definition set contains an item with constructor *constructor*, then throw a "NotSupportedError" DOMException.
5. Let *localName* be *name*.
6. Let *extends* be *options*["extends"] if it exists; otherwise null.
7. If *extends* is not null:
    1. If this's is scoped is true, then throw a "NotSupportedError" DOMException.
    2. If *extends* is a valid custom element name, then throw a "NotSupportedError" DOMException.
    3. 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 "NotSupportedError" DOMException.
    4. Set *localName* to *extends*.
8. If this's element definition is running is true, then throw a "NotSupportedError" DOMException.
9. Set this's element definition is running to true.
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 steps while catching any exceptions:
    1. Let *prototype* be ? Get(*constructor*, "prototype").
    2. If *prototype* is not an Object, then throw a TypeError exception.
    3. Let *lifecycleCallbacks* be the ordered map «["connectedCallback" → null, "disconnectedCallback" → null,

"connectedMoveCallback" → null, "adoptedCallback" → null, "attributeChangedCallback" → null]».

4. For each *callbackName* of the keys of *lifecycleCallbacks*:

    1. Let *callbackValue* be ? Get(*prototype*, *callbackName*).

    2. If *callbackValue* is not undefined, then set *lifecycleCallbacks*[*callbackName*] to the result of converting *callbackValue* to the Web IDL `Function` callback type.

5. If *lifecycleCallbacks*["attributeChangedCallback"] is not null:

    1. Let *observedAttributesIterable* be ? Get(*constructor*, "observedAttributes").

    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").

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. If *disabledFeatures* contains "internals", then set *disableInternals* to true.

10. If *disabledFeatures* contains "shadow", then set *disableShadow* to true.

11. Let *formAssociatedValue* be ? Get(*constructor*, "formAssociated").

12. Set *formAssociated* to the result of converting *formAssociatedValue* to a `boolean`.

13. If *formAssociated* is true, then for each *callbackName* of « "formAssociatedCallback", "formResetCallback", "formDisabledCallback", "formStateRestoreCallback" »:

    1. Let *callbackValue* be ? Get(*prototype*, *callbackName*).

    2. If *callbackValue* is not undefined, then set *lifecycleCallbacks*[*callbackName*] to the result of converting *callbackValue* to the Web IDL `Function` callback type.

Then, regardless of whether the above steps threw an exception or not: set this's element definition is running[p799] to false.

Finally, if the steps threw an exception, rethrow that exception.

15. Let *definition* be a new custom element definition[p798] with name[p798] *name*, local name[p798] *localName*, constructor[p798] *constructor*, observed attributes[p798] *observedAttributes*, lifecycle callbacks[p798] *lifecycleCallbacks*, form-associated[p798] *formAssociated*, disable internals[p798] *disableInternals*, and disable shadow[p798] *disableShadow*.

16. Append *definition* to this's custom element definition set[p799].

17. If this's is scoped[p799] is true, then for each *document* of this's scoped document set[p799]: upgrade particular elements within a document[p801] given this, *document*, *definition*, and *localName*.

18. Otherwise, upgrade particular elements within a document[p801] given this, this's relevant global object[p1151]'s associated `Document`[p966], *definition*, *localName*, and *name*.

19. If this's when-defined promise map[p799][*name*] exists:

    1. Resolve this's when-defined promise map[p799][*name*] with *constructor*.

    2. Remove this's when-defined promise map[p799][*name*].

To **upgrade particular elements within a document** given a `CustomElementRegistry`[p799] object *registry*, a `Document`[p135] object *document*, a custom element definition[p798] *definition*, a string *localName*, and optionally a string *name* (default *localName*):

1. Let *upgradeCandidates* be all elements that are shadow-including descendants of *document*, whose custom element registry is *registry*, whose namespace is the HTML namespace, and whose local name is *localName*, in shadow-including tree order. Additionally, if *name* is not *localName*, only include elements whose `is` value is equal to *name*.

2. For each element *element* of *upgradeCandidates*: enqueue a custom element upgrade reaction[p807] given *element* and *definition*.

The **get(*name*)** method steps are:

1. If this's custom element definition set[p799] contains an item with name[p798] *name*, then return that item's constructor[p798].
2. Return undefined.

The **getName(*constructor*)** method steps are:

1. If this's custom element definition set[p799] contains an item with constructor[p798] *constructor*, then return that item's name[p798].
2. Return null.

The **whenDefined(*name*)** method steps are:

1. If *name* is not a valid custom element name[p797], then return a promise rejected with a "SyntaxError" DOMException.
2. If this's custom element definition set[p799] contains an item with name[p798] *name*, then return a promise resolved with that item's constructor[p798].
3. If this's when-defined promise map[p799][*name*] does not exist, then set this's when-defined promise map[p799][*name*] to a new promise.
4. Return this's when-defined promise map[p799][*name*].

Example

The whenDefined()[p802] method can be used to avoid performing an action until all appropriate custom elements[p796] are defined. In this example, we combine it with the :defined[p821] pseudo-class to hide a dynamically-loaded article's contents until we're sure that all of the autonomous custom elements[p796] it uses are defined.

```
articleContainer.hidden = true;

fetch(articleURL)
  .then(response => response.text())
  .then(text => {
    articleContainer.innerHTML = text;

    return Promise.all(
      [...articleContainer.querySelectorAll(":not(:defined)")]
        .map(el => customElements.whenDefined(el.localName))
    );
  })
  .then(() => {
    articleContainer.hidden = false;
  });
```

The **upgrade(*root*)** method steps are:

1. For each shadow-including inclusive descendant *candidate* of *root*, in shadow-including tree order:
    1. If *candidate* is not an Element node, then continue.
    2. If *candidate*'s custom element registry is not this, then continue.
    3. Try to upgrade[p805] *candidate*.

Example

The upgrade()[p802] 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.

```
const el = document.createElement("spider-man");

class SpiderMan extends HTMLElement {}
customElements.define("spider-man", SpiderMan);
```

```
console.assert(!(el instanceof SpiderMan)); // not yet upgraded

customElements.upgrade(el);
console.assert(el instanceof SpiderMan);    // upgraded!
```

The **`initialize(root)`** method steps are:

1. If this's is scoped[p799] is false and either *root* is a `Document`[p135] node or *root*'s node document's custom element registry is not this, then throw a "`NotSupportedError`" `DOMException`.
2. If *root* is a `Document`[p135] node whose custom element registry is null, then set *root*'s custom element registry to this.
3. Otherwise, if *root* is a `ShadowRoot` node whose custom element registry is null, then set *root*'s custom element registry to this.
4. For each inclusive descendant *inclusiveDescendant* of *root*, in tree order:
    1. If *inclusiveDescendant* is not an `Element` node, then continue.
    2. If *inclusiveDescendant*'s custom element registry is null:
        1. Set *inclusiveDescendant*'s custom element registry to this.
        2. If this's is scoped[p799] is true, then append *inclusiveDescendant*'s node document to this's scoped document set[p799].
    3. If *inclusiveDescendant*'s custom element registry is not this, then continue.
    4. Try to upgrade[p805] *inclusiveDescendant*.

Note

Once the custom element registry of a node is initialized to a `CustomElementRegistry`[p799] object, it intentionally cannot be changed any further. This simplifies reasoning about code and allows implementations to optimize.

4.13.5 Upgrades §[p803]

To **upgrade an element**, given as input a custom element definition[p798] *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:

```
<!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);
  }
})
</script>
```

This step will thus bail out the algorithm early when upgrade an element[p803] 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[p805]. For now, we set it to "failed" so that any reentrant invocations will hit the above early-exit step[p803].

4. For each *attribute* in *element*'s attribute list, in order, enqueue a custom element callback reaction[p807] with *element*, callback name "attributeChangedCallback", and « *attribute*'s local name, null, *attribute*'s value, *attribute*'s namespace ».
5. If *element* is connected, then enqueue a custom element callback reaction[p807] with *element*, callback name "connectedCallback", and « ».
6. Add *element* to the end of *definition*'s construction stack[p798].
7. Let *C* be *definition*'s constructor[p798].
8. Let *previousRegistry* be the surrounding agent's active custom element constructor map[p798][*C*] with default null.
9. Set the surrounding agent's active custom element constructor map[p798][*C*] to *element*'s custom element registry.
10. Run the following steps while catching any exceptions:

    1. If *definition*'s disable shadow[p798] 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[p798] while attachInternals()[p809] 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[p794] uses an API decorated with the [CEReactions][p808] 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[p149] object from the constructor.

Then, perform the following steps, regardless of whether the above steps threw an exception or not:

1. If *previousRegistry* is null, then remove the surrounding agent's active custom element constructor map[p798][*C*].
2. Otherwise, set the surrounding agent's active custom element constructor map[p798][*C*] to *previousRegistry*.
3. Remove the last entry from the end of *definition*'s construction stack[p798].

Note

Assuming C calls `super()` *(as it will if it is conformant[p794]), and that the call succeeds, this will be the* already constructed *marker[p798] that replaced the* element *we pushed at the beginning of this algorithm. (The HTML element constructor[p151] carries out this replacement.)*

If C does not call `super()` *(i.e. it is not conformant[p794]), or if any step in the HTML element constructor[p151] throws, then this entry will still be* element.

Finally, if the above steps threw an exception:

1. Set *element*'s custom element definition to null.
2. Empty *element*'s custom element reaction queue[p806].
3. Rethrow the exception (thus terminating this algorithm).

Note

If the above steps threw an exception, then element's *custom element state will remain* `"failed"` *or* `"precustomized"`.

11. If *element* is a form-associated custom element[p797]:
    1. Reset the form owner[p627] of *element*. If *element* is associated with a `form`[p532] element, then enqueue a custom element callback reaction[p807] with *element*, callback name `"formAssociatedCallback"`, and « the associated `form`[p532] ».
    2. If *element* is disabled[p630], then enqueue a custom element callback reaction[p807] with *element*, callback name `"formDisabledCallback"`, and « true ».
12. Set *element*'s custom element state to `"custom"`.

To **try to upgrade an element** given an element *element*:

1. Let *definition* be the result of looking up a custom element definition[p798] given *element*'s custom element registry, *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[p807] given *element* and *definition*.

4.13.6 Custom element reactions §[p805]

A custom element[p796] possesses the ability to respond to certain occurrences by running author code:

- When upgraded[p803], its constructor[p796] is run, with no arguments.
- When it becomes connected[p47], its `connectedCallback` is called, with no arguments.
- When it becomes disconnected[p47], its `disconnectedCallback` is called, with no arguments.
- When it is moved, its `connectedMoveCallback` 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[p627] of a form-associated custom element[p797] 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[p797] is reset[p666], its `formResetCallback` is called.
- When the disabled[p630] state of a form-associated custom element[p797] is changed, its `formDisabledCallback` is called, given the new state as an argument.
- When the user agent updates a form-associated custom element[p797]'s value on behalf of a user or as part of navigation[p1105], its `formStateRestoreCallback` is called, given the new state and a string indicating a reason, `"autocomplete"` or `"restore"`,

as arguments.

We call these reactions collectively **custom element reactions**.

The way in which custom element reactions[p806] 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[p806] always are invoked in the same order as their triggering actions, at least within the local context of a single custom element[p796]. (Because custom element reaction[p806] code can perform its own mutations, it is not possible to give a global ordering guarantee across multiple elements.)

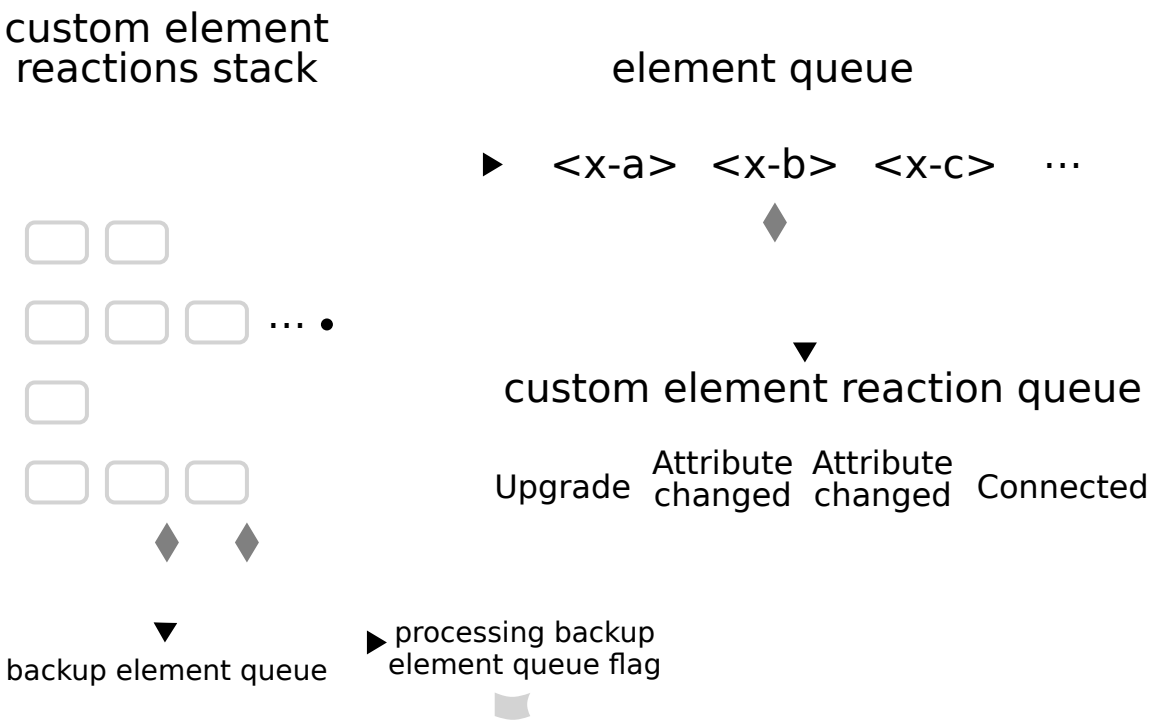
Each similar-origin window agent[p1140] has a **custom element reactions stack**, which is initially empty. A similar-origin window agent[p1140]'s **current element queue** is the element queue[p806] at the top of its custom element reactions stack[p806]. Each item in the stack is an **element queue**, which is initially empty as well. Each item in an element queue[p806] is an element. (The elements are not necessarily custom yet, since this queue is used for upgrades[p803] as well.)

Each custom element reactions stack[p806] has an associated **backup element queue**, which is an initially-empty element queue[p806]. Elements are pushed onto the backup element queue[p806] during operations that affect the DOM without going through an API decorated with `[CEReactions]`[p808], or through the parser's create an element for the token[p1418] 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[p806], each custom element reactions stack[p806] 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[p806] is of one of two types:

- An **upgrade reaction**, which will upgrade[p803] the custom element and contains a custom element definition[p798]; 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[p1141]'s custom element reactions stack[p806].
2. If *reactionsStack* is empty:
    1. Add *element* to *reactionsStack*'s backup element queue[p806].
    2. If *reactionsStack*'s processing the backup element queue[p806] flag is set, then return.
    3. Set *reactionsStack*'s processing the backup element queue[p806] flag.
    4. Queue a microtask[p1195] to perform the following steps:
        1. Invoke custom element reactions[p807] in *reactionsStack*'s backup element queue[p806].
        2. Unset *reactionsStack*'s processing the backup element queue[p806] flag.
3. Otherwise, add *element* to *element*'s relevant agent[p1141]'s current element queue[p806].

To **enqueue a custom element callback reaction**, given a custom element[p796] *element*, a callback name *callbackName*, and a list of arguments *args*, run the following steps:

1. Let *definition* be *element*'s custom element definition.
2. Let *callback* be the value of the entry in *definition*'s lifecycle callbacks[p798] with key *callbackName*.
3. If *callbackName* is "`connectedMoveCallback`" and *callback* is null:
    1. Let *disconnectedCallback* be the value of the entry in *definition*'s lifecycle callbacks[p798] with key "`disconnectedCallback`".
    2. Let *connectedCallback* be the value of the entry in *definition*'s lifecycle callbacks[p798] with key "`connectedCallback`".
    3. If *connectedCallback* and *disconnectedCallback* are null, then return.
    4. Set *callback* to the following steps:
        1. If *disconnectedCallback* is not null, then call *disconnectedCallback* with no arguments.
        2. If *connectedCallback* is not null, then call *connectedCallback* with no arguments.
4. If *callback* is null, then return.
5. If *callbackName* is "`attributeChangedCallback`":
    1. Let *attributeName* be the first element of *args*.
    2. If *definition*'s observed attributes[p798] does not contain *attributeName*, then return.
6. Add a new callback reaction[p806] to *element*'s custom element reaction queue[p806], with callback function *callback* and arguments *args*.
7. Enqueue an element on the appropriate element queue[p806] given *element*.

To **enqueue a custom element upgrade reaction**, given an element *element* and custom element definition[p798] *definition*, run the following steps:

1. Add a new upgrade reaction[p806] to *element*'s custom element reaction queue[p806], with custom element definition[p798] *definition*.
2. Enqueue an element on the appropriate element queue[p806] given *element*.

To **invoke custom element reactions** in an element queue[p806] *queue*, run the following steps:

1. While *queue* is not empty:
    1. Let *element* be the result of dequeuing from *queue*.
    2. Let *reactions* be *element*'s custom element reaction queue[p806].
    3. Repeat until *reactions* is empty:

1. Remove the first element of *reactions*, and let *reaction* be that element. Switch on *reaction*'s type:

   ↪ **upgrade reaction** p806

   Upgrade p803 *element* using *reaction*'s custom element definition p798.

   If this throws an exception, catch it, and report p1167 it for *reaction*'s custom element definition p798's constructor p798's corresponding JavaScript object's associated realm's global object p1145.

   ↪ **callback reaction** p806

   Invoke *reaction*'s callback function with *reaction*'s arguments and "`report`", and *callback this value* set to *element*.

To ensure custom element reactions p806 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 p806.

The `[CEReactions]` p808 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]` p808 extended attribute must run the following steps in place of the ones specified in their description:

1. Push a new element queue p806 onto this object's relevant agent p1141's custom element reactions stack p806.
2. Run the originally-specified steps for this construct, catching any exceptions. If the steps return a value, let *value* be the returned value. If they throw an exception, let *exception* be the thrown exception.
3. Let *queue* be the result of popping from this object's relevant agent p1141's custom element reactions stack p806.
4. Invoke custom element reactions p807 in *queue*.
5. If an exception *exception* was thrown by the original steps, rethrow *exception*.
6. If a value *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 p806 to occur).

However, in practice this imprecision could lead to non-interoperable implementations of custom element reactions p806, 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 p807 or enqueuing a custom element upgrade reaction p807, for example by modifying any attributes or child elements, must also be decorated with the `[CEReactions]` p808 attribute.

Note

As of the time of this writing, the following nonstandard or not-yet-standardized APIs are known to fall into this category:

- `HTMLInputElement` p540*'s* `webkitdirectory` *and* `incremental` *IDL attributes*
- `HTMLLinkElement` p185*'s* `scope` *IDL attribute*

4.13.7 Element internals § p808

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()` p809 method, which returns an instance of `ElementInternals` p809. The properties and methods of

ElementInternals[p809] allow control over internal features which the user agent provides to all elements.

For web developers (non-normative)

***element*.attachInternals()**[p809]

Returns an ElementInternals[p809] object targeting the custom element[p796] *element*. Throws an exception if *element* is not a custom element[p796], if the "internals" feature was disabled as part of the element definition, or if it is called twice on the same element.

Each HTMLElement[p149] has an **attached internals** (null or an ElementInternals[p809] object), initially null.

The **attachInternals()** method steps are:

1. If this's is value is not null, then throw a "NotSupportedError" DOMException.
2. Let *definition* be the result of looking up a custom element definition[p798] given this's custom element registry, this's namespace, this's local name, and null.
3. If *definition* is null, then throw a "NotSupportedError" DOMException.
4. If *definition*'s disable internals[p798] is true, then throw a "NotSupportedError" DOMException.
5. If this's attached internals[p809] is non-null, then throw a "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[p809] to a new ElementInternals[p809] instance whose target element[p810] is this.
8. Return this's attached internals[p809].

4.13.7.1 The ElementInternals[p809] interface [§p809]

The IDL for the ElementInternals[p809] 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 setFormValue((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;

  // Custom state pseudo-class
  [SameObject] readonly attribute CustomStateSet states;
};

// Accessibility semantics
```

```
ElementInternals includes ARIAMixin;

dictionary ValidityStateFlags {
  boolean valueMissing = false;
  boolean typeMismatch = false;
  boolean patternMismatch = false;
  boolean tooLong = false;
  boolean tooShort = false;
  boolean rangeUnderflow = false;
  boolean rangeOverflow = false;
  boolean stepMismatch = false;
  boolean badInput = false;
  boolean customError = false;
};
```

Each ElementInternals[p809] has a **target element**, which is a custom element[p796].

4.13.7.2 Shadow root access §[p810]

For web developers (non-normative)

***internals*.shadowRoot[p810]**

Returns the ShadowRoot for *internals*'s target element[p810], if the target element[p810] is a shadow host, or null otherwise.

The **shadowRoot** getter steps are:

1. Let *target* be this's target element[p810].
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 §[p810]

For web developers (non-normative)

***internals*.setFormValue[p811](*value*)**

Sets both the state[p811] and submission value[p811] of *internals*'s target element[p810] to *value*.

If *value* is null, the element won't participate in form submission.

***internals*.setFormValue[p811](*value*, *state*)**

Sets the submission value[p811] of *internals*'s target element[p810] to *value*, and its state[p811] to *state*.

If *value* is null, the element won't participate in form submission.

***internals*.form[p628]**

Returns the form owner[p626] of *internals*'s target element[p810].

***internals*.setValidity[p812](*flags*, *message* [, *anchor*])**

Marks *internals*'s target element[p810] 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[p810] when the form owner[p626] is validated interactively or reportValidity()[p656] is called.

***internals*.setValidity[p812]({})**

Marks *internals*'s target element[p810] as satisfying its constraints[p652].

***internals*.willValidate** p654

Returns true if *internals*'s target element p810 will be validated when the form is submitted; false otherwise.

***internals*.validity** p655

Returns the ValidityState p655 object for *internals*'s target element p810.

***internals*.validationMessage** p812

Returns the error message that would be shown to the user if *internals*'s target element p810 was to be checked for validity.

***valid* = *internals*.checkValidity()** p656

Returns true if *internals*'s target element p810 has no validity problems; false otherwise. Fires an invalid p1575 event at the element in the latter case.

***valid* = *internals*.reportValidity()** p656

Returns true if *internals*'s target element p810 has no validity problems; otherwise, returns false, fires an invalid p1575 event at the element, and (if the event isn't canceled) reports the problem to the user.

***internals*.labels** p538

Returns a NodeList of all the label p536 elements that *internals*'s target element p810 is associated with.

Each form-associated custom element p797 has **submission value**. It is used to provide one or more entries p661 on form submission. The initial value of submission value p811 is null, and submission value p811 can be null, a string, a File, or a list of entries p661.

Each form-associated custom element p797 has **state**. It is information with which the user agent can restore a user's input for the element. The initial value of state p811 is null, and state p811 can be null, a string, a File, or a list of entries p661.

The setFormValue() p811 method is used by the custom element author to set the element's submission value p811 and state p811, thus communicating these to the user agent.

When the user agent believes it is a good idea to restore a form-associated custom element p797's state p811, for example after navigation p1105 or restarting the user agent, they may enqueue a custom element callback reaction p807 with that element, callback name "`formStateRestoreCallback`", and « the state to be restored, "`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 p807 with a form-associated custom element p797, callback name "`formStateRestoreCallback`", and « the state value determined by history of state value and some heuristics, "`autocomplete`" ».

In general, the state p811 is information specified by a user, and the submission value p811 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 p797 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 p811 of the element, and "`2019-03-15`" would be a submission value p811.

Example

Suppose you develop a custom element emulating a the behavior of the existing checkbox p561 input p539 type. Its submission value p811 would be the value of its `value` content attribute, or the string "`on`". Its state p811 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 p810.
2. If *element* is not a form-associated custom element p797, then throw a "NotSupportedError" DOMException.
3. Set *element*'s submission value p811 to *value* if *value* is not a FormData object, or to a clone of *value*'s entry list otherwise.
4. If the *state* argument of the function is omitted, set *element*'s state p811 to its submission value p811.
5. Otherwise, if *state* is a FormData object, set *element*'s state p811 to a clone of *state*'s entry list.

6. Otherwise, set *element*'s state[p811] to *state*.

Each form-associated custom element[p797] has validity flags named `valueMissing`, `typeMismatch`, `patternMismatch`, `tooLong`, `tooShort`, `rangeUnderflow`, `rangeOverflow`, `stepMismatch`, and `customError`. They are false initially.

Each form-associated custom element[p797] has a **validation message** string. It is the empty string initially.

Each form-associated custom element[p797] 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[p810].
2. If *element* is not a form-associated custom element[p797], then throw a `"NotSupportedError"` `DOMException`.
3. If *message* is not given, then set it to the empty string.
4. If *flags* contains one or more true values and *message* is the empty string, then throw a `TypeError`.
5. Set *message* to the result of normalizing newlines given *message*.
6. For each entry *flag* → *value* of *flags*, set *element*'s validity flag with the name *flag* to *value*.
7. Set *element*'s validation message[p812] to the empty string if all of *element*'s validity flags are false; otherwise to *message*.
8. If *element*'s `customError` validity flag is true, then set *element*'s custom validity error message[p651] to *element*'s validation message[p812]. Otherwise, set *element*'s custom validity error message[p651] to the empty string.
9. If *anchor* is not given, then set it to *element*.
10. Otherwise, if *anchor* is not a shadow-including inclusive descendant of *element*, then throw a `"NotFoundError"` `DOMException`.
11. Set *element*'s validation anchor[p812] to *anchor*.

The **`validationMessage`** getter steps are:

1. Let *element* be this's target element[p810].
2. If *element* is not a form-associated custom element[p797], then throw a `"NotSupportedError"` `DOMException`.
3. Return *element*'s validation message[p812].

The **entry construction algorithm** for a form-associated custom element[p797], given an element *element* and an entry list[p661] *entry list*, consists of the following steps:

1. If *element*'s submission value[p811] is a list of entries[p661], then append each item of *element*'s submission value[p811] to *entry list*, and return.

   Note

   In this case, user agent does not refer to the `name`[p628] content attribute value. An implementation of form-associated custom element[p797] is responsible to decide names of entries[p661]. They can be the `name`[p628] content attribute value, they can be strings based on the `name`[p628] content attribute value, or they can be unrelated to the `name`[p628] content attribute.

2. If the element does not have a `name`[p628] attribute specified, or its `name`[p628] attribute's value is the empty string, then return.
3. If the element's submission value[p811] is not null, create an entry[p661] with the `name`[p628] attribute value and the submission value[p811], and append it to *entry list*.

4.13.7.4 Accessibility semantics §[p812]

For web developers (non-normative)

***internals*.role [= *value*]**

Sets or retrieves the default ARIA role for *internals*'s target element[p810], which will be used unless the page author overrides it using the role[p70] attribute.

***internals*.aria* [= *value*]**

Sets or retrieves various default ARIA states or property values for *internals*'s target element[p810], which will be used unless the page author overrides them using the aria-*[p70] attributes.

Each custom element[p796] has an **internal content attribute map**, which is a map, initially empty. See the Requirements related to ARIA and to platform accessibility APIs[p178] section for information on how this impacts platform accessibility APIs.

4.13.7.5 Custom state pseudo-class §[p813]

For web developers (non-normative)

***internals*.states[p813].add(*value*)**

Adds the string *value* to the element's states set[p813] to be exposed as a pseudo-class.

***internals*.states[p813].has(*value*)**

Returns true if *value* is in the element's states set[p813], otherwise false.

***internals*.states[p813].delete(*value*)**

If the element's states set[p813] has *value*, then it will be removed and true will be returned. Otherwise, false will be returned.

***internals*.states[p813].clear()**

Removes all values from the element's states set[p813].

for (const *stateName* of *internals*.states[p813])
for (const *stateName* of *internals*.states[p813].entries())
for (const *stateName* of *internals*.states[p813].keys())
for (const *stateName* of *internals*.states[p813].values())

Iterates over all values in the element's states set[p813].

***internals*.states[p813].forEach(*callback*)**

Iterates over all values in the element's states set[p813] by calling *callback* once for each value.

***internals*.states[p813].size**

Returns the number of values in the element's states set[p813].

Each custom element[p796] has a **states set**, which is a CustomStateSet[p813], initially empty.

IDL

```
[Exposed=Window]
interface CustomStateSet {
  setlike<DOMString>;
};
```

The **states** getter steps are to return this's target element[p810]'s states set[p813].

Example

The states set[p813] can expose boolean states represented by existence/non-existence of string values. If an author wants to expose a state which can have three values, it can be converted to three exclusive boolean states. For example, a state called `readyState` with `"loading"`, `"interactive"`, and `"complete"` values can be mapped to three exclusive boolean states, `"loading"`, `"interactive"`, and `"complete"`:

```
// Change the readyState from anything to "complete".
this._readyState = "complete";
this._internals.states.delete("loading");
this._internals.states.delete("interactive");
this._internals.states.add("complete");
```

4.14 Common idioms without dedicated elements §p81 4

4.14.1 Breadcrumb navigation §p81 4

This specification does not provide a machine-readable way of describing breadcrumb navigation menus. Authors are encouraged to just use a series of links in a paragraph. The nav p219 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>
  <a href="/">Main</a> ▸
  <a href="/products/">Products</a> ▸
  <a href="/products/dishwashers/">Dishwashers</a> ▸
  <a>Second hand</a>
 </p>
 <p>
  <a href="/">Main</a> ▸
  <a href="/second-hand/">Second hand</a> ▸
  <a>Dishwashers</a>
 </p>
</nav>
```

4.14.2 Tag clouds §p81 4

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 p249 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:

```
<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 class="tag-cloud-2"><a title="6 instances"  href="/t/kiwi">kiwi</a> <span>(rare)</span>
 <li class="tag-cloud-5"><a title="41 instances" href="/t/pear">pear</a> <span>(very
popular)</span>
</ul>
```

The actual frequency of each tag is given using the title p164 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 §5

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.

Example

This example demonstrates this using an extract from Abbot and Costello's famous sketch, *Who's on first*:

```
<p> Costello: Look, you gotta first baseman?
<p> Abbott: Certainly.
<p> Costello: Who's playing first?
<p> Abbott: That's right.
<p> Costello becomes exasperated.
<p> Costello: When you pay off the first baseman every month, who gets the money?
<p> Abbott: Every dollar of it.
```

Example

The following extract shows how an IM conversation log could be marked up, using the data element to provide Unix timestamps for each line. Note that the timestamps are provided in a format that the time element does not support, so the data element is used instead (namely, Unix `time_t` timestamps). Had the author wished to mark up the data using one of the date and time formats supported by the time element, that element could have been used instead of data. This could be advantageous as it would allow data analysis tools to detect the timestamps unambiguously, without coordination with the page author.

```
<p> <data value="1319898155">14:22</data> <b>egof</b> I'm not that nerdy, I've only seen 30% of the
star trek episodes
<p> <data value="1319898192">14:23</data> <b>kaj</b> if you know what percentage of the star trek
episodes you have seen, you are inarguably nerdy
<p> <data value="1319898200">14:23</data> <b>egof</b> it's unarguably
<p> <data value="1319898228">14:23</data> <i>* kaj blinks</i>
<p> <data value="1319898260">14:24</data> <b>kaj</b> you are not helping your case
```

Example

HTML does not have a good way to mark up graphs, so descriptions of interactive conversations from games are more difficult to mark up. This example shows one possible convention using dl elements to list the possible responses at each point in the conversation. Another option to consider is describing the conversation in the form of a DOT file, and outputting the result as an SVG image to place in the document. [DOT]

```
<p> Next, you meet a 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
      at this point are the same as they were after asking to borrow
      her boat, minus any options you've suggested before.
     </dl>
    </dd>
   </dl>
  </dd>
  <dt> "Vote for me in the next election!"
  <dd> <p> She turns away. The conversation finishes.
  <dt> "Madam, are you aware that your fish are running away?"
  <dd>
   <p> She looks at you skeptically and says "Fish cannot run, miss".
   <dl>
    <dt> "You got me!"
    <dd> <p> The fisher sighs and the conversation ends.
    <dt> "Only kidding."
    <dd> <p> "Good one!" she retorts. Your conversation options at this
    point are the same as those following "Hello there!" above.
    <dt> "Oh, then what are they doing?"
    <dd> <p> She looks at her fish, giving you an opportunity to steal
    her boat, which you do. The conversation ends.
   </dl>
  </dd>
 </dl>
```

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:

```
<section>
 <h1>Dialogue</h1>
 <p><small>Some characters repeat their lines in order each time you interact
 with them, others randomly pick from amongst their lines. Those who respond in
 order have numbered entries in the lists below.</small>
 <h2>The Shopkeeper</h2>
 <ul>
  <li>How may I help you?
  <li>Fresh apples!
  <li>A loaf of bread for madam?
 </ul>
 <h2>The pilot</h2>
 <p>Before the accident:
 <ul>
  <li>I'm about to fly out, sorry!
  <li>Sorry, I'm just waiting for flight clearance and then I'll be off!
 </ul>
 <p>After the accident:
```

```
 <ol>
  <li>I'm about to fly out, sorry!
  <li>Ok, I'm not leaving right now, my plane is being cleaned.
  <li>Ok, it's not being cleaned, it needs a minor repair first.
  <li>Ok, ok, stop bothering me! Truth is, I had a crash.
 </ol>
 <h2>Clan Leader</h2>
 <p>During the first clan meeting:
 <ul>
  <li>Hey, have you seen my daughter? I bet she's up to something nefarious again...
  <li>Nice weather we're having today, eh?
  <li>The name is Bailey, Jeff Bailey. How can I help you today?
  <li>A glass of water? Fresh from the well!
 </ul>
 <p>After the earthquake:
 <ol>
  <li>Everyone is safe in the shelter, we just have to put out the fire!
  <li>I'll go and tell the fire brigade, you keep hosing it down!
 </ol>
</section>
```

4.14.4 Footnotes §[p81]

HTML does not have a dedicated mechanism for marking up footnotes. Here are the suggested alternatives.

For short inline annotations, the title[p164] attribute could be used.

Example

In this example, two parts of a dialogue are annotated with footnote-like content using the title[p164] attribute.

```
<p> <b>Customer</b>: Hello! I wish to register a complaint. Hello. Miss?
<p> <b>Shopkeeper</b>: <span title="Colloquial pronunciation of 'What do you'"
>Watcha</span> mean, miss?
<p> <b>Customer</b>: Uh, I'm sorry, I have a cold. I wish to make a complaint.
<p> <b>Shopkeeper</b>: Sorry, <span title="This is, of course, a lie.">we're
closing for lunch</span>.
```

Note

Unfortunately, relying on the title[p164] 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[p164] 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[p164] attribute.

```css
[title] { border-bottom: thin dashed; }
```

For longer annotations, the a[p267] 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.

```
<p> Announcer: Number 16: The <i>hand</i>.
<p> Interviewer: Good evening. I have with me in the studio tonight
Mr Norman St John Polevaulter, who for the past few years has been
contradicting people. Mr Polevaulter, why <em>do</em> you
contradict people?
<p> Norman: I don't. <sup><a href="#fn1" id="r1">[1]</a></sup>
<p> Interviewer: You told me you did!
...
<section>
 <p id="fn1"><a href="#r1">[1]</a> This is, naturally, a lie,
 but paradoxically if it were true he could not say so without
 contradicting the interviewer and thus making it false.</p>
</section>
```

For side notes, longer annotations that apply to entire sections of the text rather than just specific words or sentences, the aside[p222] element should be used.

Example

In this example, a sidebar is given after a dialogue, giving it some context.

```
<p> <span class="speaker">Customer</span>: I will not buy this record, it is scratched.
<p> <span class="speaker">Shopkeeper</span>: I'm sorry?
<p> <span class="speaker">Customer</span>: I will not buy this record, it is scratched.
<p> <span class="speaker">Shopkeeper</span>: No no no, this's'a tobacconist's.
<aside>
 <p>In 1970, the British Empire lay in ruins, and foreign
 nationalists frequented the streets — many of them Hungarians
 (not the streets — the foreign nationals). Sadly, Alexander
 Yalt has been publishing incompetently-written phrase books.
</aside>
```

For figures or tables, footnotes can be included in the relevant figcaption[p262] or caption[p504] element, or in surrounding prose.

Example

In this example, a table has cells with footnotes that are given in prose. A figure[p259] element is used to give a single legend to the combination of the table and its footnotes.

```
<figure>
 <figcaption>Table 1. Alternative activities for knights.</figcaption>
 <table>
  <tr>
   <th> Activity
   <th> Location
   <th> Cost
  <tr>
   <td> Dance
   <td> Wherever possible
   <td> £0<sup><a href="#fn1">1</a></sup>
  <tr>
   <td> Routines, chorus scenes<sup><a href="#fn2">2</a></sup>
   <td> Undisclosed
   <td> Undisclosed
  <tr>
```

```
    <td> Dining<sup><a href="#fn3">3</a></sup>
    <td> Camelot
    <td> Cost of ham, jam, and spam<sup><a href="#fn4">4</a></sup>
  </table>
  <p id="fn1">1. Assumed.</p>
  <p id="fn2">2. Footwork impeccable.</p>
  <p id="fn3">3. Quality described as "well".</p>
  <p id="fn4">4. A lot.</p>
</figure>
```

4.15 Disabled elements §[p819]

An element is said to be **actually disabled** if it is one of the following:

- a `button`[p585] element that is disabled[p630]
- an `input`[p539] element that is disabled[p630]
- a `select`[p590] element that is disabled[p630]
- a `textarea`[p606] element that is disabled[p630]
- an `optgroup`[p600] element that has a `disabled`[p600] attribute or whose nearest ancestor `select`[p604] is disabled[p630]
- an `option`[p601] element that is disabled[p602] or whose nearest ancestor `select`[p604] is disabled[p630]
- a `fieldset`[p620] element that is a disabled fieldset[p621]
- a form-associated custom element[p797] that is disabled[p630]

Note

This definition is used to determine what elements are focusable[p876] and which elements match the `:enabled`[p822] and `:disabled`[p822] pseudo classes.

4.16 Matching HTML elements using selectors and CSS §[p819]

4.16.1 Case-sensitivity of the CSS 'attr()' function §[p819]

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][p1581]

When comparing the attribute name part of a CSS 'attr()' function to the names of namespace-less attributes on HTML elements[p46] 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 §[p819]

Selectors leaves the case-sensitivity of element names, attribute names, and attribute values to be defined by the host language. [SELECTORS][p1586]

When comparing a CSS element type selector to the names of HTML elements[p46] 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[p46] 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[p46] 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`
- `charset`
- `checked`
- `clear`
- `codetype`
- `color`
- `compact`
- `declare`
- `defer`
- `dir`
- `direction`
- `disabled`
- `enctype`
- `face`
- `frame`
- `hreflang`
- `http-equiv`
- `lang`
- `language`
- `link`
- `media`
- `method`
- `multiple`
- `nohref`
- `noresize`
- `noshade`
- `nowrap`
- `readonly`
- `rel`
- `rev`
- `rules`
- `scope`
- `scrolling`
- `selected`
- `shape`
- `target`
- `text`
- `type`
- `valign`
- `valuetype`
- `vlink`

Example

For example, the selector `[bgcolor="#ffffff"]` will match any HTML element with a `bgcolor` attribute with values including `#ffffff`, `#FFFFFF` and `#fffFFF`. This happens even if `bgcolor` has no effect for a given element (e.g., `div`[p266]).

The selector `[type=a s]` will match any HTML element with a `type` attribute whose value is `a`, but not whose value is `A`, due to the `s` flag.

All other attribute values and everything else must be treated as entirely identical to each other for the purposes of selector matching. This includes:

- IDs and classes in no-quirks mode and limited-quirks mode
- the names of elements not in the HTML namespace
- the names of HTML elements[p46] in XML documents
- the names of attributes of elements not in the HTML namespace

- the names of attributes of HTML elements[p46] in XML documents
- the names of attributes that themselves have namespaces

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.*

4.16.3 Pseudo-classes [§p821 1]

There are a number of dynamic selectors that can be used with HTML. This section defines when these selectors match HTML elements. [SELECTORS][p1586] [CSSUI][p1582]

`:defined`

The `:defined`[p821] pseudo-class must match any element that is defined.

`:link`

`:visited`

All `a`[p267] elements that have an `href`[p314] attribute, and all `area`[p489] elements that have an `href`[p314] attribute, must match one of `:link`[p821] and `:visited`[p821].

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`[p821] 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*[p821] for the purposes of defining the `:active`[p821] pseudo-class only, an HTML user agent must use the first relevant entry in the following list.

If the element is a `button`[p585] element

If the element is an `input`[p539] element whose `type`[p541] attribute is in the Submit Button[p565], Image Button[p566], Reset Button[p568], or Button[p569] state

If the element is an `a`[p267] element that has an `href`[p314] attribute

If the element is an `area`[p489] element that has an `href`[p314] attribute

If the element is focusable[p876]

The element is *being activated*[p821] if it is in a formal activation state[p821].

Example

For example, if the user is using a keyboard to push a `button`[p585] 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 being actively pointed at[p821]

The element is *being activated*[p821].

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.

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 released; for a finger in a multitouch environment, while the finger is touching the display surface).

Note

Per the definition in Selectors, `:active`[p821] *also matches flat tree ancestors of elements that are* being activated[p821]. *[SELECTORS][p1586]*

Additionally, any element that is the labeled control[p537] of a label[p536] element that is currently matching :active[p821], also matches :active[p821]. (But, it does not count as being *being activated*[p821].)

:hover

The :hover[p822] 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[p822] pseudo-class only, an HTML user agent must consider an element as being one that the user *designates*[p822] if it is an element that the user indicates using a pointing device.

Note

Per the definition in Selectors, :hover[p822] *also matches flat tree ancestors of elements that are* designated[p822]. *[SELECTORS]*[p1586]

Additionally, any element that is the labeled control[p537] of a label[p536] element that is currently matching :hover[p822], also matches :hover[p822]. (But, it does not count as being *designated*[p822].)

Example

Consider in particular a fragment such as:

```
<p> <label for=c> <input id=a> </label> <span id=b> <input id=c> </span> </p>
```

If the user designates the element with ID "a" with their pointing device, then the p[p238] element (and all its ancestors not shown in the snippet above), the label[p536] element, the element with ID "a", and the element with ID "c" will match the :hover[p822] pseudo-class. The element with ID "a" matches it by being *designated*[p822]; the label[p536] and p[p238] elements match it because of the condition in *Selectors* about flat tree ancestors; and the element with ID "c" matches it through the additional condition above on labeled controls[p537] (i.e., its label[p536] element matches :hover[p822]). However, the element with ID "b" does *not* match :hover[p822]: its flat tree descendant is not designated, even though that flat tree descendant matches :hover[p822].

:focus

For the purposes of the CSS :focus[p822] pseudo-class, an **element has the focus** when:

- it is not itself a navigable container[p1038]; and
- any of the following are true:
  - it is one of the elements listed in the current focus chain of the top-level traversable[p875]; or
  - its shadow root *shadowRoot* is not null and *shadowRoot* is the root of at least one element that has the focus[p822].

:target

For the purposes of the CSS :target[p822] pseudo-class, the Document[p135]'s *target elements* are a list containing the Document[p135]'s target element[p1104], if it is not null, or containing no elements, if it is. [SELECTORS][p1586]

:popover-open

The :popover-open[p822] pseudo-class is defined to match any HTML element[p46] whose popover[p925] attribute is not in the No Popover[p926] state and whose popover visibility state[p926] is showing[p926].

:enabled

The :enabled[p822] pseudo-class must match any button[p585], input[p539], select[p590], textarea[p606], optgroup[p600], option[p601], fieldset[p620] element, or form-associated custom element[p797] that is not actually disabled[p819].

:disabled

The :disabled[p822] pseudo-class must match any element that is actually disabled[p819].

:checked

The :checked[p822] pseudo-class must match any element falling into one of the following categories:

- input[p539] elements whose type[p541] attribute is in the Checkbox[p561] state and whose checkedness[p626] state is true
- input[p539] elements whose type[p541] attribute is in the Radio Button[p561] state and whose checkedness[p626] state is true
- option[p601] elements whose selectedness[p603] is true

:indeterminate

The :indeterminate[p823] pseudo-class must match any element falling into one of the following categories:

- input[p539] elements whose type[p541] attribute is in the Checkbox[p561] state and whose indeterminateness[p545] is true
- input[p539] elements whose type[p541] attribute is in the Radio Button[p561] state and whose radio button group[p562] contains no input[p539] elements whose checkedness[p626] state is true.
- progress[p614] elements with no value[p614] content attribute

:default

The :default[p823] pseudo-class must match any element falling into one of the following categories:

- Submit buttons[p532] that are default buttons[p657] of their form owner[p626].
- input[p539] elements to which the checked[p543] attribute applies and that have a checked[p543] attribute
- option[p601] elements that have a selected[p603] attribute

:placeholder-shown

The :placeholder-shown[p823] pseudo-class must match any element falling into one of the following categories:

- input[p539] elements that have a placeholder[p578] attribute whose value is currently being presented to the user
- textarea[p606] elements that have a placeholder[p610] attribute whose value is currently being presented to the user

:valid

The :valid[p823] pseudo-class must match any element falling into one of the following categories:

- elements that are candidates for constraint validation[p651] and that satisfy their constraints[p652]
- form[p532] elements that are not the form owner[p626] of any elements that themselves are candidates for constraint validation[p651] but do not satisfy their constraints[p652]
- fieldset[p620] elements that have no descendant elements that themselves are candidates for constraint validation[p651] but do not satisfy their constraints[p652]

:invalid

The :invalid[p823] pseudo-class must match any element falling into one of the following categories:

- elements that are candidates for constraint validation[p651] but that do not satisfy their constraints[p652]
- form[p532] elements that are the form owner[p626] of one or more elements that themselves are candidates for constraint validation[p651] but do not satisfy their constraints[p652]
- fieldset[p620] elements that have of one or more descendant elements that themselves are candidates for constraint validation[p651] but do not satisfy their constraints[p652]

:user-valid

The :user-valid[p823] pseudo-class must match input[p539], textarea[p606], and select[p590] elements whose user validity[p626] is true, are candidates for constraint validation[p651], and that satisfy their constraints[p652].

:user-invalid

The :user-invalid[p823] pseudo-class must match input[p539], textarea[p606], and select[p590] elements whose user validity[p626] is true, are candidates for constraint validation[p651] but do not satisfy their constraints[p652].

:in-range

The :in-range[p823] pseudo-class must match all elements that are candidates for constraint validation[p651], have range limitations[p574], and that are neither suffering from an underflow[p652] nor suffering from an overflow[p652].

:out-of-range

The :out-of-range[p823] pseudo-class must match all elements that are candidates for constraint validation[p651], have range limitations[p574], and that are either suffering from an underflow[p652] or suffering from an overflow[p652].

:required

The :required[p823] pseudo-class must match any element falling into one of the following categories:

- input[p539] elements that are *required*[p571]
- select[p590] elements that have a required[p591] attribute
- textarea[p606] elements that have a required[p610] attribute

:optional

The :optional[p824] pseudo-class must match any element falling into one of the following categories:

- input[p539] elements to which the required[p571] attribute applies that are not *required*[p571]
- select[p590] elements that do not have a required[p591] attribute
- textarea[p606] elements that do not have a required[p610] attribute

:autofill

:-webkit-autofill

The :autofill[p824] and :-webkit-autofill[p824] pseudo-classes must match input[p539] 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[p633] attribute, but user agents could autofill even without that attribute being involved.

:read-only

:read-write

The :read-write[p824] 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][p1586]

- input[p539] elements to which the readonly[p570] attribute applies, and that are mutable[p626] (i.e. that do not have the readonly[p570] attribute specified and that are not disabled[p630])
- textarea[p606] elements that do not have a readonly[p608] attribute, and that are not disabled[p630]
- elements that are editing hosts[p894] or editable and are neither input[p539] elements nor textarea[p606] elements

The :read-only[p824] pseudo-class must match all other HTML elements[p46].

:modal

The :modal[p824] pseudo-class must match any element falling into one of the following categories:

- dialog[p675] elements whose is modal[p680] is true
- elements whose fullscreen flag is true

:dir(ltr)

The :dir(ltr)[p824] pseudo-class must match all elements whose directionality[p168] is 'ltr[p168]'.

:dir(rtl)

The :dir(rtl)[p824] pseudo-class must match all elements whose directionality[p168] is 'rtl[p168]'.

Custom state pseudo-class

The :state(*identifier*)[p824] pseudo-class must match all custom element[p796]s whose states set[p813]'s set entries contains *identifier*.

:heading

The :heading[p824] pseudo-class must match all h1[p224], h2[p224], h3[p224], h4[p224], h5[p224], and h6[p224] elements.

:heading(*integer#*)

The :heading(*integer#*)[p824] pseudo-class must match all h1[p224], h2[p224], h3[p224], h4[p224], h5[p224], and h6[p224] elements that have a heading level[p231] of *integer*. [CSSSYNTAX][p1581] [CSSVALUES][p1581]

:playing

The :playing[p824] pseudo-class must match all media elements[p430] whose paused[p453] attribute is false.

:paused

The :paused[p825] pseudo-class must match all media elements[p430] whose paused[p453] attribute is true.

:seeking

The :seeking[p825] pseudo-class must match all media elements[p430] whose seeking[p460] attribute is true.

:buffering

The :buffering[p825] pseudo-class must match all media elements[p430] whose paused[p453] attribute is false, networkState[p435] attribute is NETWORK_LOADING[p435], and ready state is HAVE_CURRENT_DATA[p450] or less.

:stalled

The :stalled[p825] pseudo-class must match all media elements[p430] that match the :buffering[p825] pseudo-class and whose is currently stalled[p435] is true.

Note

This pseudo-class is intended for displaying player state UI when video playback has been buffering for too long. Unlike the stalled[p485] event, which fires whenever the network download has stalled regardless of playback or ready state.

:muted

The :muted[p825] pseudo-class must match all media elements[p430] that are muted[p482].

:volume-locked

The :volume-locked[p825] pseudo-class must match all media elements[p430] when the user agent's volume locked[p482] is true.

:open

The :open[p825] pseudo-class must match any element falling into one of the following categories:

- details[p666] elements that have an open[p667] attribute
- dialog[p675] elements that have an open[p677] attribute
- select[p590] elements that are a drop-down box[p1517] and whose drop-down boxes are open
- input[p539] elements that support a picker[p544] and whose pickers are open

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][p1586]*

5 Microdata §[p82]

5.1 Introduction §[p82]

5.1.1 Overview §[p82]

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 §[p82]

This section is non-normative.

At a high level, microdata consists of a group of name-value pairs. The groups are called items[p831], and each name-value pair is a property. Items and properties are represented by regular elements.

To create an item, the itemscope[p831] attribute is used.

To add a property to an item, the itemprop[p833] attribute is used on one of the item[p831]'s descendants.

Example

Here there are two items, each of which has the property "name":

```
<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:

```
<div itemscope>
 <p>My <em>name</em> is <span itemprop="name">E<strong>liz</strong>abeth</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.

Example

Here the item has three properties:

```
<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.

Example

In this example, the item has one property, "image", whose value is a URL:

```
<div itemscope>
 <img itemprop="image" src="google-logo.png" alt="Google">
</div>
```

When a string value is in some machine-readable format unsuitable for human consumption, it is expressed using the value attribute of the data element, with the human-readable version given in the element's contents.

Example

Here, there is an item with a property whose value is a product ID. The ID is not human-friendly, so the product's name is used the human-visible text instead of the ID.

```
<h1 itemscope>
 <data itemprop="product-id" value="9678AOU879">The Instigator 2000</data>
</h1>
```

For numeric data, the meter element and its value attribute can be used instead.

Example

Here a rating is given using a meter element.

```
<div itemscope itemtype="http://schema.org/Product">
 <span itemprop="name">Panasonic White 60L Refrigerator</span>
 <img src="panasonic-fridge-60l-white.jpg" alt="">
  <div itemprop="aggregateRating"
       itemscope itemtype="http://schema.org/AggregateRating">
   <meter itemprop="ratingValue" min=0 value=3.5 max=5>Rated 3.5/5</meter>
   (based on <span itemprop="reviewCount">11</span> customer reviews)
  </div>
</div>
```

Similarly, for date- and time-related data, the time element and its datetime attribute can be used instead.

Example

In this example, the item has one property, "birthday", whose value is a date:

```
<div itemscope>
 I was born on <time itemprop="birthday" datetime="2009-05-10">May 10th 2009</time>.
</div>
```

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[p836].

Example

In this example, the outer item represents a person, and the inner one represents a band:

```
<div itemscope>
 <p>Name: <span itemprop="name">Amanda</span></p>
 <p>Band: <span itemprop="band" itemscope> <span itemprop="name">Jazz Band</span> (<span
itemprop="size">12</span> players)</span></p>
</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".

The outer item in this example is a top-level microdata item.

Properties that are not descendants of the element with the `itemscope`[p831] attribute can be associated with the item[p831] using the `itemref`[p832] attribute. This attribute takes a list of IDs of elements to crawl in addition to crawling the children of the element with the `itemscope`[p831] attribute.

Example

This example is the same as the previous one, but all the properties are separated from their items[p831]:

```
<div itemscope id="amanda" itemref="a b"></div>
<p id="a">Name: <span itemprop="name">Amanda</span></p>
<div id="b" itemprop="band" itemscope itemref="c"></div>
<div id="c">
 <p>Band: <span itemprop="name">Jazz Band</span></p>
 <p>Size: <span itemprop="size">12</span> players</p>
</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 item[p831] can have multiple properties with the same name and different values.

Example

This example describes an ice cream, with two flavors:

```
<div itemscope>
 <p>Flavors in my favorite ice cream:</p>
 <ul>
  <li itemprop="flavor">Lemon sorbet</li>
  <li itemprop="flavor">Apricot sorbet</li>
 </ul>
</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":

```
<div itemscope>
 <span itemprop="favorite-color favorite-fruit">orange</span>
</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:

```
<figure>
 <img src="castle.jpeg">
 <figcaption><span itemscope><span itemprop="name">The Castle</span></span> (1986)</figcaption>
</figure>
```

```
<span itemscope><meta itemprop="name" content="The Castle"></span>
<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 §[p829]

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 re-used. 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[p831] 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[p831] is given as the value of an `itemtype`[p831] attribute on the same element as the `itemscope`[p831] attribute.

Example

Here, the item's type is "https://example.org/animals#cat":

```
<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 §[p830]

This section is non-normative.

Sometimes, an item[p831] 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`[p831] attribute) can be designed such that items[p831] get associated with their global identifier in an unambiguous way by expressing the global identifiers as URLs given in an `itemid`[p832] attribute.

The exact meaning of the URLs given in `itemid`[p832] attributes depends on the vocabulary used.

Example

Here, an item is talking about a particular book:

```
<dl itemscope
    itemtype="https://vocab.example.net/book"
    itemid="urn:isbn:0-330-34032-8">
 <dt>Title
 <dd itemprop="title">The Reality Dysfunction
 <dt>Author
 <dd itemprop="author">Peter F. Hamilton
 <dt>Publication date
 <dd><time itemprop="pubdate" datetime="1996-01-26">26 January 1996</time>
</dl>
```

The "`https://vocab.example.net/book`" vocabulary in this example would define that the `itemid`[p832] attribute takes a `urn:` URL pointing to the ISBN of the book.

5.1.5 Selecting names when defining vocabularies §[p830]

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.

```
<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
```

```
  shorthair, with a fluffy <span
  itemprop="https://example.com/color">black</span> fur with <span
  itemprop="https://example.com/color">white</span> paws and belly.</p>
  <img itemprop="img" src="hedral.jpeg" alt="" title="Hedral, age 18 months">
 </section>
```

This example has one item with the type "https://example.org/animals#cat" and the following properties:

Property	Value
name	Hedral
https://example.com/fn	Hedral
desc	Hedral is a male American domestic shorthair, with a fluffy black fur with white paws and belly.
https://example.com/color	black
https://example.com/color	white
img	.../hedral.jpeg

5.2 Encoding microdata §[p83]

5.2.1 The microdata model §[p83]

The microdata model consists of groups of name-value pairs known as items[p831].

Each group is known as an item[p831]. Each item[p831] can have item types[p831], a global identifier[p832] (if the vocabulary specified by the item types[p831] support global identifiers for items[p832]), and a list of name-value pairs. Each name in the name-value pair is known as a property[p836], and each property[p836] has one or more values[p835]. Each value[p835] is either a string or itself a group of name-value pairs (an item[p831]). The names are unordered relative to each other, but if a particular name has multiple values, they do have a relative order.

5.2.2 Items §[p83]

Every HTML element[p46] may have an **itemscope** attribute specified. The itemscope[p831] attribute is a boolean attribute[p79].

An element with the itemscope[p831] attribute specified creates a new **item**, a group of name-value pairs.

Elements with an itemscope[p831] attribute may have an **itemtype** attribute specified, to give the item types[p831] of the item[p831].

The itemtype[p831] attribute, if specified, must have a value that is an unordered set of unique space-separated tokens[p99], 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 **item types** of an item[p831] are the tokens obtained by splitting the element's itemtype attribute's value on ASCII whitespace. If the itemtype[p831] attribute is missing or parsing it in this way finds no tokens, the item[p831] is said to have no item types[p831].

The item types[p831] must all be types defined in applicable specifications[p77] and must all be defined to use the same vocabulary.

Except if otherwise specified by that specification, the URLs given as the item types[p831] should not be automatically dereferenced.

Note

A specification could define that its item type[p831] 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.

Item types[p831] are opaque identifiers, and user agents must not dereference unknown item types[p831], or otherwise deconstruct them, in order to determine how to process items[p831] that use them.

The itemtype[p831] attribute must not be specified on elements that do not have an itemscope[p831] attribute specified.

An item[p831] is said to be a **typed item** when either it has an item type[p831], or it is the value[p835] of a property[p836] of a typed item[p832]. The **relevant types** for a typed item[p832] is the item[p831]'s item types[p831], if it has any, or else is the relevant types[p832] of the item[p831] for which it is a property[p836]'s value[p835].

Elements with an itemscope[p831] attribute and an itemtype[p831] 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[p831], so that it can be related to other items[p831] on pages elsewhere on the web.

The itemid[p832] attribute, if specified, must have a value that is a valid URL potentially surrounded by spaces[p100].

The **global identifier** of an item[p831] is the value of its element's itemid[p832] attribute, if it has one, parsed[p101] relative to the node document of the element on which the attribute is specified. If the itemid[p832] attribute is missing or if parsing it returns failure, it is said to have no global identifier[p832].

The itemid[p832] attribute must not be specified on elements that do not have both an itemscope[p831] attribute and an itemtype[p831] attribute specified, and must not be specified on elements with an itemscope[p831] attribute whose itemtype[p831] attribute specifies a vocabulary that does not support global identifiers for items[p832], as defined by that vocabulary's specification.

The exact meaning of a global identifier[p832] 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[p831] 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[p831].

The itemref[p832] attribute, if specified, must have a value that is an unordered set of unique space-separated tokens[p99] none of which are identical to another token and consisting of IDs of elements in the same tree.

The itemref[p832] attribute must not be specified on elements that do not have an itemscope[p831] attribute specified.

Note

The itemref[p832] 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[p831], while keeping the properties in the cells.

Example

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[p831]:

https://md.example.com/loco
: Rolling stock with an engine

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[p831] 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:

```
<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:

```
<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
 turnouts. <meta itemprop="scale" content="HO">
</dl>
```

A passenger car with no lighting might be marked up as:

```
<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 §[p833]

Every HTML element[p46] may have an `itemprop`[p833] attribute specified, if doing so adds one or more properties[p836] to one or more items[p831] (as defined below).

The `itemprop`[p833] attribute, if specified, must have a value that is an unordered set of unique space-separated tokens[p99] 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[p832]: a **defined property name** allowed in this situation according to the specification that defines the relevant types[p832] 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[p832]: 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[p834] 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`[p833] attribute adds a property[p836] to multiple items[p831], the requirement above regarding the tokens applies for each item[p831] individually.

The **property names** of an element are the tokens that the element's `itemprop`[p833] 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[p831], 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[p836].

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:

```
<div itemscope>
 <p itemprop="a">1</p>
 <p itemprop="a">2</p>
 <p itemprop="b">test</p>
</div>
```

Thus, the following is equivalent:

```
<div itemscope>
 <p itemprop="b">test</p>
 <p itemprop="a">1</p>
 <p itemprop="a">2</p>
</div>
```

As is the following:

```
<div itemscope>
 <p itemprop="a">1</p>
 <p itemprop="b">test</p>
 <p itemprop="a">2</p>
</div>
```

And the following:

```
<div id="x">
 <p itemprop="a">1</p>
</div>
<div itemscope itemref="x">
 <p itemprop="b">test</p>
 <p itemprop="a">2</p>
</div>
```

5.2.4 Values §[p835]

The **property value** of a name-value pair added by an element with an `itemprop`[p833] attribute is as given for the first matching case in the following list:

↪ **If the element also has an `itemscope`[p831] attribute**

The value is the item[p831] created by the element.

↪ **If the element is a `meta`[p196] element**

The value is the value of the element's `content`[p197] attribute, if any, or the empty string if there is no such attribute.

↪ **If the element is an `audio`[p425], `embed`[p413], `iframe`[p404], `img`[p359], `source`[p355], `track`[p427], or `video`[p420] element**

The value is the result of encoding-parsing-and-serializing a URL[p101] given the element's `src` attribute's value, relative to the element's node document, at the time the attribute is set, or the empty string if there is no such attribute or the result is failure.

↪ **If the element is an `a`[p267], `area`[p489], or `link`[p184] element**

The value is the result of encoding-parsing-and-serializing a URL[p101] given the element's `href` attribute's value, relative to the element's node document, at the time the attribute is set, or the empty string if there is no such attribute or the result is failure.

↪ **If the element is an `object`[p416] element**

The value is the result of encoding-parsing-and-serializing a URL[p101] given the element's `data` attribute's value, relative to the element's node document, at the time the attribute is set, or the empty string if there is no such attribute or the result is failure.

↪ **If the element is a `data`[p289] element**

The value is the value of the element's `value`[p289] attribute, if it has one, or the empty string otherwise.

↪ **If the element is a `meter`[p616] element**

The value is the value of the element's `value`[p617] attribute, if it has one, or the empty string otherwise.

↪ **If the element is a `time`[p290] element**

The value is the element's datetime value[p290].

↪ **Otherwise**

The value is the element's descendant text content.

The **URL property elements** are the `a`[p267], `area`[p489], `audio`[p425], `embed`[p413], `iframe`[p404], `img`[p359], `link`[p184], `object`[p416], `source`[p355], `track`[p427], and `video`[p420] elements.

If a property's value[p835], as defined by the property's definition, is an absolute URL, the property must be specified using a URL property element[p835].

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.

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 a[p267] 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 a[p267] element (or one of the other URL property elements[p835]), because of the requirement above.

5.2.5 Associating names with items §[p836]

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[p836].

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[p832] 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[p836]; continue.
    3. Add *current* to *memory*.
    4. If *current* does not have an itemscope[p831] attribute, then add all the child elements of *current* to *pending*.
    5. If *current* has an itemprop[p833] attribute specified and has one or more property names[p834], then add *current* to *results*.
6. Sort *results* in tree order.
7. Return *results*.

A document must not contain any items[p831] for which the algorithm to find the properties of an item[p836] finds any **microdata errors**.

An item[p831] is a **top-level microdata item** if its element does not have an itemprop[p833] attribute.

All itemref[p832] attributes in a Document[p135] must be such that there are no cycles in the graph formed from representing each item[p831] in the Document[p135] as a node in the graph and each property[p836] of an item whose value[p835] is another item as an edge in the graph connecting those two items.

A document must not contain any elements that have an itemprop[p833] attribute that would not be found to be a property of any of the items[p831] in that document were their properties[p836] all to be determined.

Example

In this example, a single license statement is applied to two works, using itemref[p832] from the items representing the works:

```
<!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>
  <figure itemscope itemtype="http://n.whatwg.org/work" itemref="licenses">
   <img itemprop="work" src="images/mailbox.jpeg" alt="Outside the house is a mailbox. It has a
leaflet inside.">
   <figcaption itemprop="title">The mailbox.</figcaption>
  </figure>
  <footer>
   <p id="licenses">All images licensed under the <a itemprop="license"
   href="http://www.opensource.org/licenses/mit-license.php">MIT
   license</a>.</p>
  </footer>
 </body>
</html>
```

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`

5.2.6 Microdata and other namespaces §[p837]

Currently, the `itemscope`[p831], `itemprop`[p833], and other microdata attributes are only defined for HTML elements[p46]. 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 §[p837]

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 §[p838]

An item with the item type[p831] **`http://microformats.org/profile/hcard`** represents a person's or organization's contact information.

This vocabulary does not support global identifiers for items[p832].

The following are the type's defined property names[p834]. 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. [RFC6350][p1586]

`kind`

Describes what kind of contact the item represents.

The value[p835] must be text that is identical to one of the kind strings[p845].

A single property with the name `kind`[p838] may be present within each item[p831] with the type `http://microformats.org/profile/hcard`[p838].

`fn`

Gives the formatted text corresponding to the name of the person or organization.

The value[p835] must be text.

Exactly one property with the name `fn`[p838] must be present within each item[p831] with the type `http://microformats.org/profile/hcard`[p838].

`n`

Gives the structured name of the person or organization.

The value[p835] must be an item[p831] with zero or more of each of the `family-name`[p838], `given-name`[p838], `additional-name`[p838], `honorific-prefix`[p838], and `honorific-suffix`[p839] properties.

Exactly one property with the name `n`[p838] must be present within each item[p831] with the type `http://microformats.org/profile/hcard`[p838].

`family-name` (inside `n`[p838])

Gives the family name of the person, or the full name of the organization.

The value[p835] must be text.

Any number of properties with the name `family-name`[p838] may be present within the item[p831] that forms the value[p835] of the `n`[p838] property of an item[p831] with the type `http://microformats.org/profile/hcard`[p838].

`given-name` (inside `n`[p838])

Gives the given-name of the person.

The value[p835] must be text.

Any number of properties with the name `given-name`[p838] may be present within the item[p831] that forms the value[p835] of the `n`[p838] property of an item[p831] with the type `http://microformats.org/profile/hcard`[p838].

`additional-name` (inside `n`[p838])

Gives the any additional names of the person.

The value[p835] must be text.

Any number of properties with the name `additional-name`[p838] may be present within the item[p831] that forms the value[p835] of the `n`[p838] property of an item[p831] with the type `http://microformats.org/profile/hcard`[p838].

`honorific-prefix` (inside `n`[p838])

Gives the honorific prefix of the person.

The value[p835] must be text.

Any number of properties with the name `honorific-prefix`[p838] may be present within the item[p831] that forms the value[p835] of the

n[p838] property of an item[p831] with the type http://microformats.org/profile/hcard[p838].

honorific-suffix (inside n[p838])

Gives the honorific suffix of the person.

The value[p835] must be text.

Any number of properties with the name honorific-suffix[p839] may be present within the item[p831] that forms the value[p835] of the n[p838] property of an item[p831] with the type http://microformats.org/profile/hcard[p838].

nickname

Gives the nickname of the person or organization.

Note

The nickname is the descriptive name given instead of or in addition to the one belonging to a person, place, or thing. It can also be used to specify a familiar form of a proper name specified by the fn[p838] or n[p838] properties.

The value[p835] must be text.

Any number of properties with the name nickname[p839] may be present within each item[p831] with the type http://microformats.org/profile/hcard[p838].

photo

Gives a photograph of the person or organization.

The value[p835] must be an absolute URL.

Any number of properties with the name photo[p839] may be present within each item[p831] with the type http://microformats.org/profile/hcard[p838].

bday

Gives the birth date of the person or organization.

The value[p835] must be a valid date string[p87].

A single property with the name bday[p839] may be present within each item[p831] with the type http://microformats.org/profile/hcard[p838].

anniversary

Gives the birth date of the person or organization.

The value[p835] must be a valid date string[p87].

A single property with the name anniversary[p839] may be present within each item[p831] with the type http://microformats.org/profile/hcard[p838].

sex

Gives the biological sex of the person.

The value[p835] must be one of F, meaning "female", M, meaning "male", N, meaning "none or not applicable", O, meaning "other", or U, meaning "unknown".

A single property with the name sex[p839] may be present within each item[p831] with the type http://microformats.org/profile/hcard[p838].

gender-identity

Gives the gender identity of the person.

The value[p835] must be text.

A single property with the name gender-identity[p839] may be present within each item[p831] with the type http://microformats.org/profile/hcard[p838].

adr

Gives the delivery address of the person or organization.

The value[p835] must be an item[p831] with zero or more type[p840], post-office-box[p840], extended-address[p840], and street-address[p840] properties, and optionally a locality[p840] property, optionally a region[p840] property, optionally a postal-code[p841] property, and optionally a country-name[p841] property.

If no type[p840] properties are present within an item[p831] that forms the value[p835] of an adr[p840] property of an item[p831] with the type http://microformats.org/profile/hcard[p838], then the address type string[p845] work[p845] is implied.

Any number of properties with the name adr[p840] may be present within each item[p831] with the type http://microformats.org/profile/hcard[p838].

type (inside adr[p840])

Gives the type of delivery address.

The value[p835] must be text that is identical to one of the address type strings[p845].

Any number of properties with the name type[p840] may be present within the item[p831] that forms the value[p835] of an adr[p840] property of an item[p831] with the type http://microformats.org/profile/hcard[p838], but within each such adr[p840] property item[p831] there must only be one type[p840] property per distinct value.

post-office-box (inside adr[p840])

Gives the post office box component of the delivery address of the person or organization.

The value[p835] must be text.

Any number of properties with the name post-office-box[p840] may be present within the item[p831] that forms the value[p835] of an adr[p840] property of an item[p831] with the type http://microformats.org/profile/hcard[p838].

Note

vCard *urges authors not to use this field.*

extended-address (inside adr[p840])

Gives an additional component of the delivery address of the person or organization.

The value[p835] must be text.

Any number of properties with the name extended-address[p840] may be present within the item[p831] that forms the value[p835] of an adr[p840] property of an item[p831] with the type http://microformats.org/profile/hcard[p838].

Note

vCard *urges authors not to use this field.*

street-address (inside adr[p840])

Gives the street address component of the delivery address of the person or organization.

The value[p835] must be text.

Any number of properties with the name street-address[p840] may be present within the item[p831] that forms the value[p835] of an adr[p840] property of an item[p831] with the type http://microformats.org/profile/hcard[p838].

locality (inside adr[p840])

Gives the locality component (e.g. city) of the delivery address of the person or organization.

The value[p835] must be text.

A single property with the name locality[p840] may be present within the item[p831] that forms the value[p835] of an adr[p840] property of an item[p831] with the type http://microformats.org/profile/hcard[p838].

region (inside adr[p840])

Gives the region component (e.g. state or province) of the delivery address of the person or organization.

The value[p835] must be text.

A single property with the name region[p840] may be present within the item[p831] that forms the value[p835] of an adr[p840] property of an item[p831] with the type http://microformats.org/profile/hcard[p838].

postal-code (inside adr[p840])

Gives the postal code component of the delivery address of the person or organization.

The value[p835] must be text.

A single property with the name postal-code[p841] may be present within the item[p831] that forms the value[p835] of an adr[p840] property of an item[p831] with the type http://microformats.org/profile/hcard[p838].

country-name (inside adr[p840])

Gives the country name component of the delivery address of the person or organization.

The value[p835] must be text.

A single property with the name country-name[p841] may be present within the item[p831] that forms the value[p835] of an adr[p840] property of an item[p831] with the type http://microformats.org/profile/hcard[p838].

tel

Gives the telephone number of the person or organization.

The value[p835] 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[p831] with zero or more type[p841] properties and exactly one value[p841] property. [E163][p1582] [X121][p1588]

If no type[p841] properties are present within an item[p831] that forms the value[p835] of a tel[p841] property of an item[p831] with the type http://microformats.org/profile/hcard[p838], or if the value[p835] of such a tel[p841] property is text, then the telephone type string[p845] voice[p845] is implied.

Any number of properties with the name tel[p841] may be present within each item[p831] with the type http://microformats.org/profile/hcard[p838].

type (inside tel[p841])

Gives the type of telephone number.

The value[p835] must be text that is identical to one of the telephone type strings[p845].

Any number of properties with the name type[p841] may be present within the item[p831] that forms the value[p835] of a tel[p841] property of an item[p831] with the type http://microformats.org/profile/hcard[p838], but within each such tel[p841] property item[p831] there must only be one type[p841] property per distinct value.

value (inside tel[p841])

Gives the actual telephone number of the person or organization.

The value[p835] must be text that can be interpreted as a telephone number as defined in the CCITT specifications E.163 and X.121. [E163][p1582] [X121][p1588]

Exactly one property with the name value[p841] must be present within the item[p831] that forms the value[p835] of a tel[p841] property of an item[p831] with the type http://microformats.org/profile/hcard[p838].

email

Gives the email address of the person or organization.

The value[p835] must be text.

Any number of properties with the name email[p841] may be present within each item[p831] with the type http://microformats.org/profile/hcard[p838].

impp

Gives a URL for instant messaging and presence protocol communications with the person or organization.

The value[p835] must be an absolute URL.

Any number of properties with the name impp[p841] may be present within each item[p831] with the type http://microformats.org/profile/hcard[p838].

lang

Gives a language understood by the person or organization.

The value[p835] must be a valid BCP 47 language tag. [BCP47][p1579]

Any number of properties with the name lang[p842] may be present within each item[p831] with the type http://microformats.org/profile/hcard[p838].

tz

Gives the time zone of the person or organization.

The value[p835] 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[p81] 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[p81] that is exactly two digits long and that represents a number in the range 00..59.

Any number of properties with the name tz[p842] may be present within each item[p831] with the type http://microformats.org/profile/hcard[p838].

geo

Gives the geographical position of the person or organization.

The value[p835] 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.

Any number of properties with the name geo[p842] may be present within each item[p831] with the type http://microformats.org/profile/hcard[p838].

title

Gives the job title, functional position or function of the person or organization.

The value[p835] must be text.

Any number of properties with the name title[p842] may be present within each item[p831] with the type http://microformats.org/profile/hcard[p838].

role

Gives the role, occupation, or business category of the person or organization.

The value[p835] must be text.

Any number of properties with the name role[p842] may be present within each item[p831] with the type http://microformats.org/profile/hcard[p838].

logo

Gives the logo of the person or organization.

The value[p835] must be an absolute URL.

Any number of properties with the name logo[p843] may be present within each item[p831] with the type http://microformats.org/profile/hcard[p838].

agent

Gives the contact information of another person who will act on behalf of the person or organization.

The value[p835] must be either an item[p831] with the type http://microformats.org/profile/hcard[p838], or an absolute URL, or text.

Any number of properties with the name agent[p843] may be present within each item[p831] with the type http://microformats.org/profile/hcard[p838].

org

Gives the name and units of the organization.

The value[p835] must be either text or an item[p831] with one organization-name[p843] property and zero or more organization-unit[p843] properties.

Any number of properties with the name org[p843] may be present within each item[p831] with the type http://microformats.org/profile/hcard[p838].

organization-name (inside org[p843])

Gives the name of the organization.

The value[p835] must be text.

Exactly one property with the name organization-name[p843] must be present within the item[p831] that forms the value[p835] of an org[p843] property of an item[p831] with the type http://microformats.org/profile/hcard[p838].

organization-unit (inside org[p843])

Gives the name of the organization unit.

The value[p835] must be text.

Any number of properties with the name organization-unit[p843] may be present within the item[p831] that forms the value[p835] of the org[p843] property of an item[p831] with the type http://microformats.org/profile/hcard[p838].

member

Gives a URL that represents a member of the group.

The value[p835] must be an absolute URL.

Any number of properties with the name member[p843] may be present within each item[p831] with the type http://microformats.org/profile/hcard[p838] if the item[p831] also has a property with the name kind[p838] whose value is "group[p845]".

related

Gives a relationship to another entity.

The value[p835] must be an item[p831] with one url[p843] property and one rel[p844] properties.

Any number of properties with the name related[p843] may be present within each item[p831] with the type http://microformats.org/profile/hcard[p838].

url (inside related[p843])

Gives the URL for the related entity.

The value[p835] must be an absolute URL.

Exactly one property with the name url[p843] must be present within the item[p831] that forms the value[p835] of a related[p843] property of an item[p831] with the type http://microformats.org/profile/hcard[p838].

rel (inside related[p843])

Gives the relationship between the entity and the related entity.

The value[p835] must be text that is identical to one of the relationship strings[p845].

Exactly one property with the name rel[p844] must be present within the item[p831] that forms the value[p835] of a related[p843] property of an item[p831] with the type http://microformats.org/profile/hcard[p838].

categories

Gives the name of a category or tag that the person or organization could be classified as.

The value[p835] must be text.

Any number of properties with the name categories[p844] may be present within each item[p831] with the type http://microformats.org/profile/hcard[p838].

note

Gives supplemental information or a comment about the person or organization.

The value[p835] must be text.

Any number of properties with the name note[p844] may be present within each item[p831] with the type http://microformats.org/profile/hcard[p838].

rev

Gives the revision date and time of the contact information.

The value[p835] must be text that is a valid global date and time string[p92].

Note

The value distinguishes the current revision of the information for other renditions of the information.

Any number of properties with the name rev[p844] may be present within each item[p831] with the type http://microformats.org/profile/hcard[p838].

sound

Gives a sound file relating to the person or organization.

The value[p835] must be an absolute URL.

Any number of properties with the name sound[p844] may be present within each item[p831] with the type http://microformats.org/profile/hcard[p838].

uid

Gives a globally unique identifier corresponding to the person or organization.

The value[p835] must be text.

A single property with the name uid[p844] may be present within each item[p831] with the type http://microformats.org/profile/hcard[p838].

url

Gives a URL relating to the person or organization.

The value[p835] must be an absolute URL.

Any number of properties with the name url[p844] may be present within each item[p831] with the type http://microformats.org/profile/hcard[p838].

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.

contact
acquaintance
friend
met
worker
colleague
resident
neighbor
child
parent
sibling
spouse
kin
muse
crush
date
sweetheart
me

Has the meaning defined in XFN. [XFN][p1588]

5.3.1.1 Conversion to vCard [§][p846]

Given a list of nodes *nodes* in a `Document`[p135], 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[p831] with the item type[p831] `http://microformats.org/profile/hcard`[p838], then there is no vCard. Abort the algorithm, returning nothing.
2. Let *node* be the first node in *nodes* that is an item[p831] with the item type[p831] `http://microformats.org/profile/hcard`[p838].
3. Let *output* be an empty string.
4. Add a vCard line[p848] with the type "`BEGIN`" and the value "`VCARD`" to *output*.
5. Add a vCard line[p848] with the type "`PROFILE`" and the value "`VCARD`" to *output*.
6. Add a vCard line[p848] with the type "`VERSION`" and the value "`4.0`" to *output*.
7. Add a vCard line[p848] with the type "`SOURCE`" and the result of escaping the vCard text string[p849] that is the document's URL as the value to *output*.
8. If the `title` element[p143] is not null, add a vCard line[p848] with the type "`NAME`" and with the result of escaping the vCard text string[p849] obtained from the `title` element[p143]'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[p836] *node*: for each name *name* in *element*'s property names[p834], 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[p835] is an item[p831] *subitem* and *name* is `n`[p838]

       1. Let *value* be the empty string.
       2. Append to *value* the result of collecting the first vCard subproperty[p849] named `family-name`[p838] in *subitem*.

3. Append a U+003B SEMICOLON character (;) to *value*.
4. Append to *value* the result of collecting the first vCard subproperty[p849] named `given-name`[p838] in *subitem*.
5. Append a U+003B SEMICOLON character (;) to *value*.
6. Append to *value* the result of collecting the first vCard subproperty[p849] named `additional-name`[p838] in *subitem*.
7. Append a U+003B SEMICOLON character (;) to *value*.
8. Append to *value* the result of collecting the first vCard subproperty[p849] named `honorific-prefix`[p838] in *subitem*.
9. Append a U+003B SEMICOLON character (;) to *value*.
10. Append to *value* the result of collecting the first vCard subproperty[p849] named `honorific-suffix`[p839] in *subitem*.

If the property's value[p835] is an item[p831] *subitem* and *name* is `adr`[p840]

1. Let *value* be the empty string.
2. Append to *value* the result of collecting vCard subproperties[p849] named `post-office-box`[p840] in *subitem*.
3. Append a U+003B SEMICOLON character (;) to *value*.
4. Append to *value* the result of collecting vCard subproperties[p849] named `extended-address`[p840] in *subitem*.
5. Append a U+003B SEMICOLON character (;) to *value*.
6. Append to *value* the result of collecting vCard subproperties[p849] named `street-address`[p840] in *subitem*.
7. Append a U+003B SEMICOLON character (;) to *value*.
8. Append to *value* the result of collecting the first vCard subproperty[p849] named `locality`[p840] in *subitem*.
9. Append a U+003B SEMICOLON character (;) to *value*.
10. Append to *value* the result of collecting the first vCard subproperty[p849] named `region`[p840] in *subitem*.
11. Append a U+003B SEMICOLON character (;) to *value*.
12. Append to *value* the result of collecting the first vCard subproperty[p849] named `postal-code`[p841] in *subitem*.
13. Append a U+003B SEMICOLON character (;) to *value*.
14. Append to *value* the result of collecting the first vCard subproperty[p849] named `country-name`[p841] in *subitem*.
15. If there is a property named `type`[p840] in *subitem*, and the first such property has a value[p835] that is not an item[p831] and whose value consists only of ASCII alphanumerics, then add a parameter named "TYPE" whose value is the value[p835] of that property to *parameters*.

If the property's value[p835] is an item[p831] *subitem* and *name* is `org`[p843]

1. Let *value* be the empty string.
2. Append to *value* the result of collecting the first vCard subproperty[p849] named `organization-name`[p843] in *subitem*.
3. For each property named `organization-unit`[p843] in *subitem*, run the following steps:
    1. If the value[p835] of the property is an item[p831], then skip this property.

2. Append a U+003B SEMICOLON character (;) to *value*.

3. Append the result of escaping the vCard text string[p849] given by the value[p835] of the property to *value*.

If the property's value[p835] is an item[p831] *subitem* with the item type[p831] `http://microformats.org/profile/hcard`[p838] and *name* is `related`[p843]

1. Let *value* be the empty string.

2. If there is a property named `url`[p843] in *subitem*, and its element is a URL property element[p835], then append the result of escaping the vCard text string[p849] given by the value[p835] 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`[p844] in *subitem*, and the first such property has a value[p835] that is not an item[p831] and whose value consists only of ASCII alphanumerics, then add a parameter named "`RELATION`" whose value is the value[p835] of that property to *parameters*.

If the property's value[p835] is an item[p831] and *name* is none of the above

1. Let *value* be the result of collecting the first vCard subproperty[p849] named `value` in *subitem*.

2. If there is a property named `type` in *subitem*, and the first such property has a value[p835] that is not an item[p831] and whose value consists only of ASCII alphanumerics, then add a parameter named "`TYPE`" whose value is the value[p835] of that property to *parameters*.

If the property's value[p835] is not an item[p831] and *name* is `sex`[p839]

If this is the first such property to be found, set *sex* to the property's value[p835].

If the property's value[p835] is not an item[p831] and *name* is `gender-identity`[p839]

If this is the first such property to be found, set *gender-identity* to the property's value[p835].

Otherwise (the property's value[p835] is not an item[p831])

1. Let *value* be the property's value[p835].

2. If *element* is one of the URL property elements[p835], add a parameter with the name "`VALUE`" and the value "`URI`" to *parameters*.

3. Otherwise, if *name* is `bday`[p839] or `anniversary`[p839] and the *value* is a valid date string[p87], add a parameter with the name "`VALUE`" and the value "`DATE`" to *parameters*.

4. Otherwise, if *name* is `rev`[p844] and the *value* is a valid global date and time string[p92], 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`[p842], 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[p848] 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[p848] 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[p848] 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 *line* be an empty string.
2. Append *type*, converted to ASCII uppercase, to *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 *line*.
    2. Append the parameter's name to *line*.
    3. Append a U+003D EQUALS SIGN character (=) to *line*.
    4. Append the parameter's value to *line*.
4. Append a U+003A COLON character (:) to *line*.
5. Append *value* to *line*.
6. Let *maximum length* be 75.
7. 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.
8. Append (what remains of) *line* to *output*.
9. Append a U+000D CARRIAGE RETURN character (CR) to *output*.
10. Append a U+000A LINE FEED character (LF) to *output*.

When the steps above require the user agent to obtain the result of **collecting vCard subproperties** named *subname* in *subitem*, the user agent must run the following steps:

1. Let *value* be the empty string.
2. For each property named *subname* in the item *subitem*, run the following substeps:
    1. If the value[p835] of the property is itself an item[p831], then skip this property.
    2. If this is not the first property named *subname* in *subitem* (ignoring any that were skipped by the previous step), then append a U+002C COMMA character (,) to *value*.
    3. Append the result of escaping the vCard text string[p849] given by the value[p835] of the property to *value*.
3. Return *value*.

When the steps above require the user agent to obtain the result of **collecting the first vCard subproperty** named *subname* in *subitem*, the user agent must run the following steps:

1. If there are no properties named *subname* in *subitem*, then return the empty string.
2. If the value[p835] of the first property named *subname* in *subitem* is an item[p831], then return the empty string.
3. Return the result of escaping the vCard text string[p849] given by the value[p835] of the first property named *subname* in *subitem*.

When the above algorithms say the user agent is to **escape the vCard text string** *value*, the user agent must use the following steps:

1. Prefix every U+005C REVERSE SOLIDUS character (\) in *value* with another U+005C REVERSE SOLIDUS character (\).
2. Prefix every U+002C COMMA character (,) in *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[p838] *item type*[p831] *and defined property names*[p834].

5.3.1.2 Examples §[p850]

This section is non-normative.

Example

Here is a long example vCard for a fictional character called "Jack Bauer":

```
<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" itemscope>
  <span itemprop="organization-name">Counter-Terrorist Unit</span>
  (<span itemprop="organization-unit">Los Angeles Division</span>)
 </p>
 <p>
  <span itemprop="adr" itemscope>
   <span itemprop="street-address">10201 W. Pico Blvd.</span><br>
   <span itemprop="locality">Los Angeles</span>,
   <span itemprop="region">CA</span>
   <span itemprop="postal-code">90064</span><br>
   <span itemprop="country-name">United States</span><br>
  </span>
  <span itemprop="geo">34.052339;-118.410623</span>
 </p>
 <h2>Assorted Contact Methods</h2>
 <ul>
  <li itemprop="tel" itemscope>
   <span itemprop="value">+1 (310) 597 3781</span> <span itemprop="type">work</span>
   <meta itemprop="type" content="voice">
  </li>
  <li><a itemprop="url" 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>
  <li><a itemprop="url" href="http://www.jackbauerfacts.com/">Jack Bauer Facts</a></li>
  <li itemprop="email"><a
href="mailto:j.bauer@la.ctu.gov.invalid">j.bauer@la.ctu.gov.invalid</a></li>
  <li itemprop="tel" itemscope>
   <span itemprop="value">+1 (310) 555 3781</span> <span>
   <meta itemprop="type" content="cell">mobile phone</span>
  </li>
 </ul>
 <ins datetime="2008-07-20 21:00:00+01:00">
```

```
  <meta itemprop="rev" content="2008-07-20 21:00:00+01:00">
  <p itemprop="tel" itemscope><strong>Update!</strong>
  My new <span itemprop="type">home</span> phone number is
  <span itemprop="value">01632 960 123</span>.</p>
 </ins>
</section>
```

The odd line wrapping is needed because newlines are meaningful in microdata: newlines would be preserved in a conversion to, for example, the vCard format.

Example

This example shows a site's contact details (using the address[p230] element) containing an address with two street components:

```
<address itemscope itemtype="http://microformats.org/profile/hcard">
 <strong itemprop="fn"><span itemprop="n" itemscope><span itemprop="given-name">Alfred</span>
 <span itemprop="family-name">Person</span></span></strong> <br>
 <span itemprop="adr" itemscope>
  <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>
 </span>
</address>
```

Example

The vCard vocabulary can be used to just mark up people's names:

```
<span itemscope itemtype="http://microformats.org/profile/hcard"
><span itemprop=fn><span itemprop="n" itemscope><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 §[p85] 1

An item with the item type[p831] **http://microformats.org/profile/hcalendar#vevent** represents an event.

This vocabulary does not support global identifiers for items[p832].

The following are the type's defined property names[p834]. 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][p1585]

Note

Only the parts of the iCalendar vocabulary relating to events are used here; this vocabulary cannot express a complete iCalendar instance.

attach

Gives the address of an associated document for the event.

The value[p835] must be an absolute URL.

Any number of properties with the name attach[p852] may be present within each item[p831] with the type http://microformats.org/profile/hcalendar#vevent[p851].

categories

Gives the name of a category or tag that the event could be classified as.

The value[p835] must be text.

Any number of properties with the name categories[p852] may be present within each item[p831] with the type http://microformats.org/profile/hcalendar#vevent[p851].

class

Gives the access classification of the information regarding the event.

The value[p835] 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[p852] may be present within each item[p831] with the type http://microformats.org/profile/hcalendar#vevent[p851].

comment

Gives a comment regarding the event.

The value[p835] must be text.

Any number of properties with the name comment[p852] may be present within each item[p831] with the type http://microformats.org/profile/hcalendar#vevent[p851].

description

Gives a detailed description of the event.

The value[p835] must be text.

A single property with the name description[p852] may be present within each item[p831] with the type http://microformats.org/profile/hcalendar#vevent[p851].

geo

Gives the geographical position of the event.

The value[p835] 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[p852] may be present within each item[p831] with the type http://microformats.org/profile/hcalendar#vevent[p851].

location

Gives the location of the event.

The value[p835] must be text.

A single property with the name location[p853] may be present within each item[p831] with the type http://microformats.org/profile/hcalendar#vevent[p851].

resources

Gives a resource that will be needed for the event.

The value[p835] must be text.

Any number of properties with the name resources[p853] may be present within each item[p831] with the type http://microformats.org/profile/hcalendar#vevent[p851].

status

Gives the confirmation status of the event.

The value[p835] must be text with one of the following values:

- tentative
- confirmed
- canceled

A single property with the name status[p853] may be present within each item[p831] with the type http://microformats.org/profile/hcalendar#vevent[p851].

summary

Gives a short summary of the event.

The value[p835] must be text.

User agents should replace U+000A LINE FEED (LF) characters in the value[p835] by U+0020 SPACE characters when using the value.

A single property with the name summary[p853] may be present within each item[p831] with the type http://microformats.org/profile/hcalendar#vevent[p851].

dtend

Gives the date and time by which the event ends.

If the property with the name dtend[p853] is present within an item[p831] with the type http://microformats.org/profile/hcalendar#vevent[p851] that has a property with the name dtstart[p854] whose value is a valid date string[p87], then the value[p835] of the property with the name dtend[p853] must be text that is a valid date string[p87] also. Otherwise, the value[p835] of the property must be text that is a valid global date and time string[p92].

In either case, the value[p835] be later in time than the value of the dtstart[p854] property of the same item[p831].

Note

The time given by the dtend[p853] property is not inclusive. For day-long events, therefore, the dtend[p853] property's value[p835] will be the day after the end of the event.

A single property with the name dtend[p853] may be present within each item[p831] with the type http://microformats.org/profile/

hcalendar#vevent[p851], so long as that http://microformats.org/profile/hcalendar#vevent[p851] does not have a property with the name duration[p854].

dtstart

Gives the date and time at which the event starts.

The value[p835] must be text that is either a valid date string[p87] or a valid global date and time string[p92].

Exactly one property with the name dtstart[p854] must be present within each item[p831] with the type http://microformats.org/profile/hcalendar#vevent[p851].

duration

Gives the duration of the event.

The value[p835] must be text that is a valid vevent duration string[p855].

The duration represented is the sum of all the durations represented by integers in the value.

A single property with the name duration[p854] may be present within each item[p831] with the type http://microformats.org/profile/hcalendar#vevent[p851], so long as that http://microformats.org/profile/hcalendar#vevent[p851] does not have a property with the name dtend[p853].

transp

Gives whether the event is to be considered as consuming time on a calendar, for the purpose of free-busy time searches.

The value[p835] must be text with one of the following values:

- opaque
- transparent

A single property with the name transp[p854] may be present within each item[p831] with the type http://microformats.org/profile/hcalendar#vevent[p851].

contact

Gives the contact information for the event.

The value[p835] must be text.

Any number of properties with the name contact[p854] may be present within each item[p831] with the type http://microformats.org/profile/hcalendar#vevent[p851].

url

Gives a URL for the event.

The value[p835] must be an absolute URL.

A single property with the name url[p854] may be present within each item[p831] with the type http://microformats.org/profile/hcalendar#vevent[p851].

uid

Gives a globally unique identifier corresponding to the event.

The value[p835] must be text.

A single property with the name uid[p854] may be present within each item[p831] with the type http://microformats.org/profile/hcalendar#vevent[p851].

exdate

Gives a date and time at which the event does not occur despite the recurrence rules.

The value[p835] must be text that is either a valid date string[p87] or a valid global date and time string[p92].

Any number of properties with the name exdate[p854] may be present within each item[p831] with the type http://microformats.org/profile/hcalendar#vevent[p851].

rdate

Gives a date and time at which the event recurs.

The value[p835] must be text that is one of the following:

- A valid date string[p87].
- A valid global date and time string[p92].
- A valid global date and time string[p92] followed by a U+002F SOLIDUS character (/) followed by a second valid global date and time string[p92] representing a later time.
- A valid global date and time string[p92] followed by a U+002F SOLIDUS character (/) followed by a valid vevent duration string[p855].

Any number of properties with the name rdate[p855] may be present within each item[p831] with the type http://microformats.org/profile/hcalendar#vevent[p851].

rrule

Gives a rule for finding dates and times at which the event occurs.

The value[p835] must be text that matches the RECUR value type defined in *iCalendar*. [RFC5545][p1585]

A single property with the name rrule[p855] may be present within each item[p831] with the type http://microformats.org/profile/hcalendar#vevent[p851].

created

Gives the date and time at which the event information was first created in a calendaring system.

The value[p835] must be text that is a valid global date and time string[p92].

A single property with the name created[p855] may be present within each item[p831] with the type http://microformats.org/profile/hcalendar#vevent[p851].

last-modified

Gives the date and time at which the event information was last modified in a calendaring system.

The value[p835] must be text that is a valid global date and time string[p92].

A single property with the name last-modified[p855] may be present within each item[p831] with the type http://microformats.org/profile/hcalendar#vevent[p851].

sequence

Gives a revision number for the event information.

The value[p835] must be text that is a valid non-negative integer[p81].

A single property with the name sequence[p855] may be present within each item[p831] with the type http://microformats.org/profile/hcalendar#vevent[p851].

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[p81] 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[p81] 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[p81] 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[p81] 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[p81] 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 [p856 §6]

Given a list of nodes *nodes* in a `Document`[p135], a user agent must run the following algorithm to **extract any vEvent data represented by those nodes**:

1. If none of the nodes in *nodes* are items[p831] with the type `http://microformats.org/profile/hcalendar#vevent`[p851], then there is no vEvent data. Abort the algorithm, returning nothing.
2. Let *output* be an empty string.
3. Add an iCalendar line[p857] with the type "`BEGIN`" and the value "`VCALENDAR`" to *output*.
4. Add an iCalendar line[p857] with the type "`PRODID`" and the value equal to a user-agent-specific string representing the user agent to *output*.
5. Add an iCalendar line[p857] with the type "`VERSION`" and the value "`2.0`" to *output*.
6. For each node *node* in *nodes* that is an item[p831] with the type `http://microformats.org/profile/hcalendar#vevent`[p851], run the following steps:
    1. Add an iCalendar line[p857] with the type "`BEGIN`" and the value "`VEVENT`" to *output*.
    2. Add an iCalendar line[p857] 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 *output*. [RFC5545][p1585]
    3. For each element *element* that is a property of the item[p836] *node*: for each name *name* in *element*'s property names[p834], run the appropriate set of substeps from the following list:

        If the property's value[p835] is an item[p831]

        Skip the property.

        If the property is `dtend`[p853]
        If the property is `dtstart`[p854]
        If the property is `exdate`[p854]
        If the property is `rdate`[p855]
        If the property is `created`[p855]
        If the property is `last-modified`[p855]

        Let *value* be the result of stripping all U+002D HYPHEN-MINUS (-) and U+003A COLON (:) characters from the property's value[p835].

        If the property's value[p835] is a valid date string[p87], then add an iCalendar line[p857] with the type *name* and the value *value* to *output*, with the annotation "`VALUE=DATE`".

        Otherwise, if the property's value[p835] is a valid global date and time string[p92], then add an iCalendar line[p857] with the type *name* and the value *value* to *output*, with the annotation "`VALUE=DATE-TIME`".

        Otherwise, skip the property.

        Otherwise

        Add an iCalendar line[p857] with the type *name* and the property's value[p835] to *output*.
    4. Add an iCalendar line[p857] with the type "`END`" and the value "`VEVENT`" to *output*.

7. Add an iCalendar line[p857] 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[p851] *item type*[p831] *and defined property names*[p834].

5.3.2.2 Examples §[p857]

This section is non-normative.

Example

Here is an example of a page that uses the vEvent vocabulary to mark up an event:

```
<body itemscope itemtype="http://microformats.org/profile/hcalendar#vevent">
 ...
```

```
 <h1 itemprop="summary">Bluesday Tuesday: Money Road</h1>
 ...
 <time itemprop="dtstart" datetime="2009-05-05T19:00:00Z">May 5th @ 7pm</time>
 (until <time itemprop="dtend" datetime="2009-05-05T21:00:00Z">9pm</time>)
 ...
 <a href="http://livebrum.co.uk/2009/05/05/bluesday-tuesday-money-road"
    rel="bookmark" itemprop="url">Link to this page</a>
 ...
 <p>Location: <span itemprop="location">The RoadHouse</span></p>
 ...
 <p><input type=button value="Add to Calendar"
           onclick="location = getCalendar(this)"></p>
 ...
 <meta itemprop="description" content="via livebrum.co.uk">
</body>
```

The `getCalendar()` function is left as an exercise for the reader.

The same page could offer some markup, such as the following, for copy-and-pasting into blogs:

```
<div itemscope itemtype="http://microformats.org/profile/hcalendar#vevent">
 <p>I'm going to
 <strong itemprop="summary">Bluesday Tuesday: Money Road</strong>,
 <time itemprop="dtstart" datetime="2009-05-05T19:00:00Z">May 5th at 7pm</time>
 to <time itemprop="dtend" datetime="2009-05-05T21:00:00Z">9pm</time>,
 at <span itemprop="location">The RoadHouse</span>!</p>
 <p><a href="http://livebrum.co.uk/2009/05/05/bluesday-tuesday-money-road"
       itemprop="url">See this event on livebrum.co.uk</a>.</p>
 <meta itemprop="description" content="via livebrum.co.uk">
</div>
```

5.3.3 Licensing works §[p858]

An item with the item type[p831] **`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[p834].

`work`

Identifies the work being described.

The value[p835] must be an absolute URL.

Exactly one property with the name `work`[p858] must be present within each item[p831] with the type `http://n.whatwg.org/work`[p858].

`title`

Gives the name of the work.

A single property with the name `title`[p858] may be present within each item[p831] with the type `http://n.whatwg.org/work`[p858].

`author`

Gives the name or contact information of one of the authors or creators of the work.

The value[p835] must be either an item[p831] with the type `http://microformats.org/profile/hcard`[p838], or text.

Any number of properties with the name `author`[p858] may be present within each item[p831] with the type `http://n.whatwg.org/work`[p858].

`license`

Identifies one of the licenses under which the work is available.

The value[p835] must be an absolute URL.

Any number of properties with the name `license`[p859] may be present within each item[p831] with the type `http://n.whatwg.org/work`[p858].

5.3.3.1 Examples §[p859]

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.

```
<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 §[p859]

5.4.1 JSON §[p859]

Given a list of nodes *nodes* in a `Document`[p135], 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[p836], and if it is then get the object[p859] 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][p1583]

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[p831], add an entry to *result* called "`type`" whose value is an array listing the item types[p831] of *item*, in the order they were specified on the `itemtype`[p831] attribute.
5. If the *item* has a global identifier[p832], add an entry to *result* called "`id`" whose value is the global identifier[p832] of *item*.
6. Let *properties* be an empty object.
7. For each element *element* that has one or more property names[p834] and is one of the properties of the item[p836] *item*, in the order those elements are given by the algorithm that returns the properties of an item[p836], run the following substeps:
    1. Let *value* be the property value[p835] of *element*.
    2. If *value* is an item[p831], then: if *value* is in *memory*, then let *value* be the string "`ERROR`". Otherwise, get the object[p859] 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[p834], 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:

```
<!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="?comments=0">
 </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>
 <section>
  <h1>Comments</h1>
  <article itemprop="comment" 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 itemprop="commentTime" datetime="2013-08-29">15 minutes ago</time></p>
   </footer>
   <p>Ha!</p>
  </article>
  <article itemprop="comment" 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 itemprop="commentTime" datetime="2013-08-29">5 minutes ago</time></p>
   </footer>
   <p>When you say "we got it down"...</p>
```

```
  </article>
 </section>
</article>
```

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`):

```
{
  "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 §p862

6.1 The hidden attribute §p862

All HTML elements may have the **hidden** content attribute set. The hidden attribute is an enumerated attribute with the following keywords and states:

Keyword	State	Brief description
hidden	**Hidden**	Will not be rendered.
until-found	**Hidden Until Found**	Will not be rendered, but content inside will be accessible to find-in-page and fragment navigation.

The attribute's *missing value default* is the **Not Hidden** state, and its *invalid value default* and *empty value default* are both the Hidden state.

When an element has the hidden attribute in the Hidden state, 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 are in the Hidden state.

Note

The requirement for user agents not to render elements that are in the Hidden state can be implemented indirectly through the style layer. For example, a web browser could implement these requirements using the rules suggested in the Rendering section.

When an element has the hidden attribute in the Hidden Until Found state, it indicates that the element is hidden like the Hidden state but the content inside the element will be accessible to find-in-page and fragment navigation. When these features attempt to scroll to a target which is in the element's subtree, the user agent will remove the hidden attribute in order to reveal the content before scrolling to it by running the ancestor revealing algorithm on the target node.

Note

Web browsers will use 'content-visibility: hidden' instead of 'display: none' when the hidden attribute is in the Hidden Until Found state, as specified 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 state. Authors therefore have to take care when writing their style sheets to make sure that the attribute is still styled as expected. In addition, legacy user agents which don't support the Hidden Until Found state will have 'display: none' instead of 'content-visibility: hidden', so authors are encouraged to make sure that their style sheets don't change the 'display' or 'content-visibility' properties of Hidden Until Found elements.

Since elements with the hidden attribute in the Hidden Until Found state use 'content-visibility: hidden' instead of 'display: none', there are two caveats of the Hidden Until Found state that make it different from the Hidden state:

1. *The element needs to be affected by layout containment in order to be revealed by find-in-page. This means that if the element in the Hidden Until Found state has a 'display' value of 'none', 'contents', or 'inline', then the element will not be revealed by find-in-page.*
2. *The element will still have a generated box when in the Hidden Until Found state, which means that borders, margin, and padding will still be rendered around the element.*

Example

In the following skeletal example, the attribute is used to hide the web game's main screen until the user logs in:

```
<h1>The Example Game</h1>
<section id="login">
 <h2>Login</h2>
```

```
  <form>
   ...
   <!-- calls login() once the user's credentials have been checked -->
  </form>
  <script>
   function login() {
     // switch screens
     document.getElementById('login').hidden = true;
     document.getElementById('game').hidden = false;
   }
  </script>
 </section>
 <section id="game" hidden>
  ...
 </section>
```

The hidden[p862] attribute must not be used to hide content that could legitimately be shown in another presentation. For example, it is incorrect to use hidden[p862] 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[p862], it is hidden from all presentations, including, for instance, screen readers.

Elements that are not themselves hidden[p862] must not hyperlink[p313] to elements that are hidden[p862]. The `for` attributes of label[p536] and output[p611] elements that are not themselves hidden[p862] must similarly not refer to elements that are hidden[p862]. In both cases, such references would cause user confusion.

Elements and scripts may, however, refer to elements that are hidden[p862] in other contexts.

Example

For example, it would be incorrect to use the href[p314] attribute to link to a section marked with the hidden[p862] 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[p862]. 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.

Similarly, a canvas[p713] element with the hidden[p862] attribute could be used by a scripted graphics engine as an off-screen buffer, and a form control could refer to a hidden form[p532] element using its form[p627] attribute.

Elements in a section hidden by the hidden[p862] attribute are still active, e.g. scripts and form controls in such sections still execute and submit respectively. Only their presentation to the user changes.

The **hidden** getter steps are:

1. If the hidden[p862] attribute is in the Hidden Until Found[p862] state, then return "until-found[p862]".
2. If the hidden[p862] attribute is set, then return true.
3. Return false.

The hidden[p863] setter steps are:

1. If the given value is a string that is an ASCII case-insensitive match for "until-found[p862]", then set the hidden[p862] attribute to "until-found[p862]".
2. Otherwise, if the given value is false, then remove the hidden[p862] attribute.
3. Otherwise, if the given value is the empty string, then remove the hidden[p862] attribute.
4. Otherwise, if the given value is null, then remove the hidden[p862] attribute.
5. Otherwise, if the given value is 0, then remove the hidden[p862] attribute.

6. Otherwise, if the given value is NaN, then remove the `hidden`[p862] attribute.

7. Otherwise, set the `hidden`[p862] attribute to the empty string.

An **ancestor reveal pair** is a tuple consisting of a **node** and a **string**.

The **ancestor revealing algorithm** given a node *target* is:

1. Let *ancestorsToReveal* be « ».
2. Let *ancestor* be *target*.
3. While *ancestor* has a parent node within the flat tree:
    1. If *ancestor* has a `hidden`[p862] attribute in the Hidden Until Found[p862] state, then append (*ancestor*, "`until-found`") to *ancestorsToReveal*.
    2. If *ancestor* is slotted into the second slot of a `details`[p666] element which does not have an `open`[p667] attribute, then append (*ancestor*'s parent node, "`details`") to *ancestorsToReveal*.
    3. Set *ancestor* to the parent node of *ancestor* within the flat tree.
4. For each (*ancestorToReveal*, *revealType*) of *ancestorsToReveal*:
    1. If *ancestorToReveal* is not connected, then return.
    2. If *revealType* is "`until-found`":
        1. If *ancestorToReveal*'s `hidden`[p862] attribute is not in the Hidden Until Found[p862] state, then return.
        2. Fire an event named `beforematch`[p1574] at *ancestorToReveal* with the `bubbles` attribute initialized to true.
        3. If *ancestorToReveal* is not connected, then return.
        4. If *ancestorToReveal*'s `hidden`[p862] attribute is not in the Hidden Until Found[p862] state, then return.
        5. Remove the `hidden`[p862] attribute from *ancestorToReveal*.
    3. Otherwise:
        1. Assert: *revealType* is "`details`".
        2. If *ancestorToReveal* has an `open`[p667] attribute, then return.
        3. Set *ancestorToReveal*'s `open`[p667] attribute to the empty string.

6.2 Page visibility §[p864]

A traversable navigable[p1037]'s system visibility state[p1037], including its initial value upon creation, is determined by the user agent. It represents, for example, whether the browser window is minimized, a browser tab is currently in the background, or a system element such as a task switcher obscures the page.

When a user agent determines that the system visibility state[p1037] for traversable navigable[p1037] *traversable* has changed to *newState*, it must run the following steps:

1. Let *navigables* be the inclusive descendant navigables[p1041] of *traversable*'s active document[p1036].
2. For each *navigable* of *navigables* in what order?:
    1. Let *document* be *navigable*'s active document[p1036].
    2. Queue a global task[p1195] on the user interaction task source[p1204] given *document*'s relevant global object[p1151] to update the visibility state[p865] of *document* with *newState*.

A `Document`[p135] has a **visibility state**, which is either "`hidden`" or "`visible`", initially set to "`hidden`".

The **`visibilityState`** getter steps are to return this's visibility state[p864].

The **hidden** getter steps are to return true if this's visibility state[p864] is "hidden", otherwise false.

To **update the visibility state** of Document[p135] *document* to *visibilityState*:

1. If *document*'s visibility state[p864] equals *visibilityState*, then return.
2. Set *document*'s visibility state[p864] to *visibilityState*.
3. Queue a new VisibilityStateEntry[p865] whose visibility state[p866] is *visibilityState* and whose timestamp[p865] is the current high resolution time given *document*'s relevant global object[p1151].
4. Run the screen orientation change steps with *document*. [SCREENORIENTATION][p1586]
5. Run the view transition page visibility change steps with *document*.
6. Run any **page visibility change steps** which may be defined in other specifications, with visibility state[p864] and *document*.

   It would be better if specification authors sent a pull request to add calls from here into their specifications directly, instead of using the page visibility change steps[p865] hook, to ensure well-defined cross-specification call order. As of the time of this writing the following specifications are known to have page visibility change steps[p865], which will be run in an unspecified order: *Device Posture API* and *Web NFC*. [DEVICEPOSTURE][p1582] [WEBNFC][p1588]
7. Fire an event named visibilitychange[p1576] at *document*, with its bubbles attribute initialized to true.

To **set the initial visibility state** of Document[p135] *document* to *visibilityState*:

1. Set *document*'s visibility state[p864] to *visibilityState*.
2. Queue a new VisibilityStateEntry[p865] whose visibility state[p866] is *document*'s visibility state[p864] and whose timestamp[p865] is 0.

6.2.1 The VisibilityStateEntry[p865] interface §[p865]

The VisibilityStateEntry[p865] interface exposes visibility changes to the document, from the moment the document becomes active.

Example

For example, this allows JavaScript code in the page to examine correlation between visibility changes and paint timing:

```
function wasHiddenBeforeFirstContentfulPaint() {
    const fcpEntry = performance.getEntriesByName("first-contentful-paint")[0];
    const visibilityStateEntries = performance.getEntriesByType("visibility-state");
    return visibilityStateEntries.some(e =>
                                       e.startTime < fcpEntry.startTime &&
                                       e.name === "hidden");
}
```

Note

Since hiding a page can cause throttling of rendering and other user-agent operations, it is common to use visibility changes as an indication that such throttling has occurred. However, other things could also cause throttling in different browsers, such as long periods of inactivity.

IDL

```
[Exposed=(Window)]
interface VisibilityStateEntry : PerformanceEntry {
  readonly attribute DOMString name;                // shadows inherited name
  readonly attribute DOMString entryType;           // shadows inherited entryType
  readonly attribute DOMHighResTimeStamp startTime; // shadows inherited startTime
  readonly attribute unsigned long duration;        // shadows inherited duration
};
```

The VisibilityStateEntry[p865] has an associated DOMHighResTimeStamp **timestamp**.

The VisibilityStateEntry[p865] has an associated "visible" or "hidden" **visibility state**.

The **name** getter steps are to return this's visibility state[p866].

The **entryType** getter steps are to return "visibility-state".

The **startTime** getter steps are to return this's timestamp[p865].

The **duration** getter steps are to return zero.

6.3 Inert subtrees §[p866]

Note

See also inert[p866] for an explanation of the attribute of the same name.

A node (in particular elements and text nodes) can be **inert**. When a node is inert[p866]:

- Hit-testing must act as if the 'pointer-events' CSS property were set to 'none'.
- Text selection functionality must act as if the 'user-select' CSS property were set to 'none'.
- If it is editable, the node behaves as if it were non-editable.
- The user agent should ignore the node for the purposes of find-in-page[p901].

Note

Inert nodes generally cannot be focused, and user agents do not expose the inert nodes to accessibility APIs or assistive technologies. Inert nodes that are commands[p672] will become inoperable to users, in the manner described above.

User agents may allow the user to override the restrictions on find-in-page[p901] and text selection, however.

By default, a node is not inert[p866].

6.3.1 Modal dialogs and inert subtrees §[p866]

A Document[p135] *document* is **blocked by a modal dialog** *subject* if *subject* is the topmost dialog[p675] 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 flat tree descendants, must become inert[p866].

subject can additionally become inert[p866] via the inert[p866] attribute, but only if specified on *subject* itself (i.e., *subject* escapes inertness of ancestors); *subject*'s flat tree descendants can become inert[p866] in a similar fashion.

Note

The dialog[p675] element's showModal()[p679] method causes this mechanism to trigger, by adding the dialog[p675] element to its node document's top layer.

6.3.2 The inert attribute §[p866]

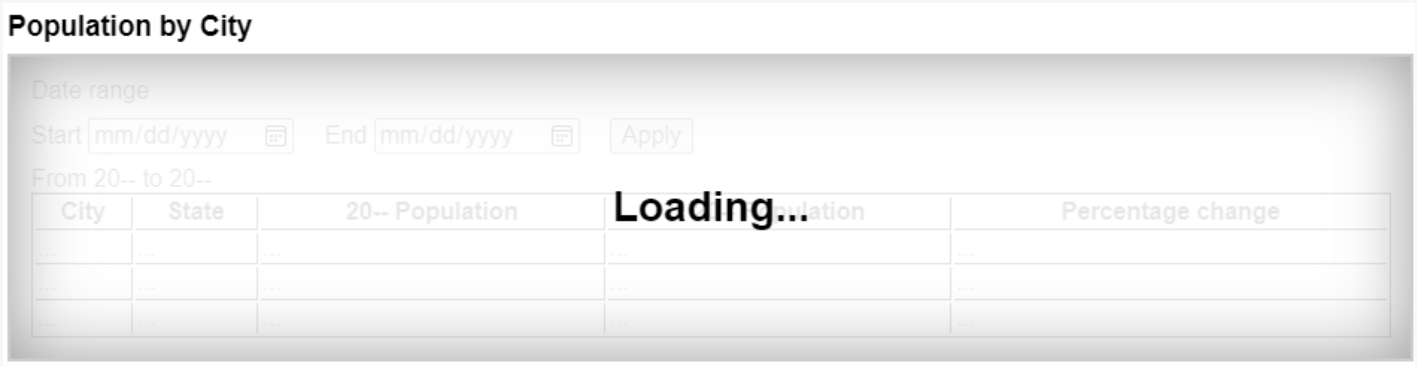
The inert[p866] attribute is a boolean attribute[p79] that indicates, by its presence, that the element and all its flat tree descendants which don't otherwise escape inertness (such as modal dialogs) are to be made inert[p866] by the user agent.

An inert subtree should not contain any content or controls which are critical to understanding or using aspects of the page which are not in the inert state. Content in an inert subtree will not be perceivable by all users, or interactive. Authors should not specify elements as inert unless the content they represent are also visually obscured in some way. In most cases, authors should not specify the inert[p866] attribute on individual form controls. In these instances, the disabled[p630] attribute is probably more appropriate.

Example

The following example shows how to mark partially loaded content, visually obscured by a "loading" message, as inert.

```
<section aria-labelledby=s1>
  <h3 id=s1>Population by City</h3>
  <div class=container>
    <div class=loading><p>Loading...</p></div>
    <div inert>
      <form>
        <fieldset>
          <legend>Date range</legend>
          <div>
            <label for=start>Start</label>
            <input type=date id=start>
          </div>
          <div>
            <label for=end>End</label>
            <input type=date id=end>
          </div>
          <div>
            <button>Apply</button>
          </div>
        </fieldset>
      </form>
      <table>
        <caption>From 20-- to 20--</caption>
        <thead>
          <tr>
            <th>City</th>
            <th>State</th>
            <th>20-- Population</th>
            <th>20-- Population</th>
            <th>Percentage change</th>
          </tr>
        </thead>
        <tbody>
          <!-- ... -->
        </tbody>
      </table>
    </div>
  </div>
</section>
```



The "loading" overlay obscures the inert content, making it visually apparent that the inert content is not presently accessible. Notice that the heading and "loading" text are not descendants of the element with the inert[p866] attribute. This will ensure this text is accessible to all users, while the inert content cannot be interacted with by anyone.

Note

By default, there is no persistent visual indication of an element or its subtree being inert. Appropriate visual styles for such content is often context-dependent. For instance, an inert off-screen navigation panel would not require a default style, as its off-screen position visually obscures the content. Similarly, a modal dialog element's backdrop will serve as the means to visually obscure the inert content of the web page, rather than styling the inert content specifically.

However, for many other situations authors are strongly encouraged to clearly mark what parts of their document are active and which are inert, to avoid user confusion. In particular, it is worth remembering that not all users can see all parts of a page at once; for example, users of screen readers, users on small devices or with magnifiers, and even users using particularly small windows might not be able to see the active part of a page and might get frustrated if inert sections are not obviously inert.

6.4 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.

6.4.1 Data model

For the purpose of tracking user activation, each Window *W* has two relevant values:

- A **last activation timestamp**, which is either a DOMHighResTimeStamp, positive infinity (indicating that *W* has never been activated), or negative infinity (indicating that the activation has been consumed). Initially positive infinity.
- A **last history-action activation timestamp**, which is either a DOMHighResTimeStamp or positive infinity, initially 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 (e.g., for opening popups).

Note

The transient activation duration 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.

We then have the following boolean user activation states for *W*:

Sticky activation

When the current high resolution time given *W* is greater than or equal to the last activation timestamp in *W*, *W* is said to have sticky activation.

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.

Transient activation

When the current high resolution time given *W* is greater than or equal to the last activation timestamp in *W*, and less than the last activation timestamp in *W* plus the transient activation duration, then *W* is said to have transient activation.

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 *W* gets.

The transient activation state is considered **expired** if it becomes false because the transient activation duration time has elapsed since the last user activation. Note that it can become false even before the expiry time through an activation consumption.

History-action activation

When the last history-action activation timestamp of *W* is not equal to the last activation timestamp of *W*, then *W* is said to have history-action activation.

This is a special variant of user activation, used to allow access to certain session history APIs which, if used too frequently, would make it harder for the user to traverse back using browser UI[p1137]. It starts with a false value, and becomes true whenever the user interacts with *W*, but is reset to false through history-action activation consumption[p869]. This ensures such APIs cannot be used multiple times in a row without an intervening user activation. But unlike transient activation[p868], there is no time limit within which such APIs must be used.

Note

The last activation timestamp[p868] and last history-action activation timestamp[p868] are retained even after the `Document`[p135] changes its fully active[p1051] status (e.g., after navigating away from a `Document`[p135], or navigating to a cached `Document`[p135]). This means sticky activation[p868] state spans multiple navigations as long as the same `Document`[p135] gets reused. For the transient activation state, the original expiry[p868] time remains unchanged (i.e., the state still expires within the transient activation duration[p868] limit from the original activation triggering input event[p869]). It is important to consider this when deciding whether to base certain things off sticky activation[p868] or transient activation[p868].

6.4.2 Processing model §[p869]

When a user interaction causes firing of an activation triggering input event[p869] in a `Document`[p135] *document*, the user agent must perform the following **activation notification** steps *before* dispatching the event:

1. Assert: *document* is fully active[p1051].
2. Let *windows* be « *document*'s relevant global object[p1151] ».
3. Extend *windows* with the active window[p1036] of each of *document*'s ancestor navigables[p1041].
4. Extend *windows* with the active window[p1036] of each of *document*'s descendant navigables[p1041], filtered to include only those navigables[p1035] whose active document[p1036]'s origin is same origin[p940] with *document*'s origin.
5. For each *window* in *windows*:
    1. Set *window*'s last activation timestamp[p868] to the current high resolution time.
    2. Notify the close watcher manager about user activation[p904] given *window*.

An **activation triggering input event** is any event whose `isTrusted` attribute is true and whose `type` is one of:

- "`keydown`", provided the key is neither the Esc key nor a shortcut key reserved by the user agent;
- "`mousedown`";
- "`pointerdown`", provided the event's `pointerType` is "`mouse`";
- "`pointerup`", provided the event's `pointerType` is not "`mouse`"; or
- "`touchend`".

Activation consuming APIs[p870] defined in this and other specifications can **consume user activation** by performing the following steps, given a `Window`[p965] *W*:

1. If *W*'s navigable[p966] is null, then return.
2. Let *top* be *W*'s navigable[p966]'s top-level traversable[p1037].
3. Let *navigables* be the inclusive descendant navigables[p1041] of *top*'s active document[p1036].
4. Let *windows* be the list of `Window`[p965] objects constructed by taking the active window[p1036] of each item in *navigables*.
5. For each *window* in *windows*, if *window*'s last activation timestamp[p868] is not positive infinity, then set *window*'s last activation timestamp[p868] to negative infinity.

History-action activation-consuming APIs[p870] can **consume history-action user activation** by performing the following steps, given a `Window`[p965] *W*:

1. If *W*'s navigable[p966] is null, then return.

2. Let *top* be *W*'s navigable[p966]'s top-level traversable[p1037].
3. Let *navigables* be the inclusive descendant navigables[p1041] of *top*'s active document[p1036].
4. Let *windows* be the list of Window[p965] objects constructed by taking the active window[p1036] of each item in *navigables*.
5. For each *window* in *windows*, set *window*'s last history-action activation timestamp[p868] to *window*'s last activation timestamp[p868].

Note

Note the asymmetry in the sets of browsing contexts[p1046] in the page that are affected by an activation notification[p869] vs an activation consumption[p869]: an activation consumption changes (to false) the transient activation[p868] 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[p870] from a single user activation (possibly by exploiting a deep hierarchy of iframe[p404]s).

6.4.3 APIs gated by user activation §[p87]

APIs that are dependent on user activation are classified into different levels:

Sticky activation-gated APIs

These APIs require the sticky activation[p868] state to be true, so they are blocked until the very first user activation.

Transient activation-gated APIs

These APIs require the transient activation[p868] state to be true, but they don't consume[p869] it, so multiple calls are allowed per user activation until the transient state expires[p868].

Transient activation-consuming APIs

These APIs require the transient activation[p868] state to be true, and they consume user activation[p869] in each call to prevent multiple calls per user activation.

History-action activation-consuming APIs

These APIs require the history-action activation[p868] state to be true, and they consume history-action user activation[p869] in each call to prevent multiple calls per user activation.

6.4.4 The UserActivation[p870] interface §[p87]

Each Window[p965] has an **associated UserActivation**, which is a UserActivation[p870] object. Upon creation of the Window[p965] object, its associated UserActivation[p870] must be set to a new UserActivation[p870] object created in the Window[p965] object's relevant realm[p1151].

IDL

```
[Exposed=Window]
interface UserActivation {
  readonly attribute boolean hasBeenActive;
  readonly attribute boolean isActive;
};

partial interface Navigator {
  [SameObject] readonly attribute UserActivation userActivation;
};
```

For web developers (non-normative)

navigator[p1262].userActivation[p871].hasBeenActive[p871]

Returns whether the window has sticky activation[p868].

navigator[p1262].userActivation[p871].isActive[p871]

Returns whether the window has transient activation[p868].

The **userActivation** getter steps are to return this's relevant global object[p1151]'s associated UserActivation[p870].

The **hasBeenActive** getter steps are to return true if this's relevant global object[p1151] has sticky activation[p868], and false otherwise.

The **isActive** getter steps are to return true if this's relevant global object[p1151] has transient activation[p868], and false otherwise.

6.4.5 User agent automation §[p87]

For the purposes of user-agent automation and application testing, this specification defines the following extension command for the *Web Driver* specification. It is optional for a user agent to support the following extension command. [WEBDRIVER][p1588]

HTTP Method	URI Template
`POST`	/session/{*session id*}/window/consume-user-activation

The remote end steps are:

1. Let *window* be the current browsing context's active window[p1046].
2. Let *consume* be true if *window* has transient activation[p868]; otherwise false.
3. If *consume* is true, then consume user activation[p869] of *window*.
4. Return success with data *consume*.

6.5 Activation behavior of elements §[p87]

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[p1218] at the element.

For web developers (non-normative)

***element*.click[p871]()**

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[p630], then return.
2. If this element's click in progress flag[p871] is set, then return.
3. Set this element's click in progress flag[p871].
4. Fire a synthetic pointer event[p1217] named `click` at this element, with the *not trusted flag* set.
5. Unset this element's click in progress flag[p871].

6.5.1 The ToggleEvent[p871] interface §[p87]

IDL

```
[Exposed=Window]
interface ToggleEvent : Event {
  constructor(DOMString type, optional ToggleEventInit eventInitDict = {});
  readonly attribute DOMString oldState;
```

```
  readonly attribute DOMString newState;
  readonly attribute Element? source;
};

dictionary ToggleEventInit : EventInit {
  DOMString oldState = "";
  DOMString newState = "";
  Element? source = null;
};
```

For web developers (non-normative)

***event*.oldState**[p872]

Set to "closed" when transitioning from closed to open, or set to "open" when transitioning from open to closed.

***event*.newState**[p872]

Set to "open" when transitioning from closed to open, or set to "closed" when transitioning from open to closed.

***event*.source**[p872]

Set to the element which initiated the toggle, which can be set up with the popovertarget[p935] and commandfor[p586] attributes. If there is no source element, then it is set to null.

The **oldState** and **newState** attributes must return the values they are initialized to.

The **source** getter steps are to return the result of retargeting source[p872] against this's currentTarget.

DOM standard issue #1328 tracks how to better standardize associated event data in a way which makes sense on Events. Currently an event attribute initialized to a value cannot also have a getter, and so an internal slot (or map of additional fields) is required to properly specify this.

A **toggle task tracker** is a struct which has:

task

A task[p1194] which fires a ToggleEvent[p871].

old state

A string which represents the task[p872]'s event's value for the oldState[p872] attribute.

6.5.2 The CommandEvent[p872] interface §[p872]

IDL

```
[Exposed=Window]
interface CommandEvent : Event {
  constructor(DOMString type, optional CommandEventInit eventInitDict = {});
  readonly attribute Element? source;
  readonly attribute DOMString command;
};

dictionary CommandEventInit : EventInit {
  Element? source = null;
  DOMString command = "";
};
```

For web developers (non-normative)

***event*.command**[p873]

Returns what action the element can take.

***event*.source**[p873]

Returns the Element that was interacted with in order to cause this event.

The **command** attribute must return the value it was initialized to.

The **source** getter steps are to return the result of retargeting `source`[p873] against this's `currentTarget`.

DOM standard issue #1328 tracks how to better standardize associated event data in a way which makes sense on Events. Currently an event attribute initialized to a value cannot also have a getter, and so an internal slot (or map of additional fields) is required to properly specify this.

6.6 Focus §[p873]

6.6.1 Introduction §[p873]

This section is non-normative.

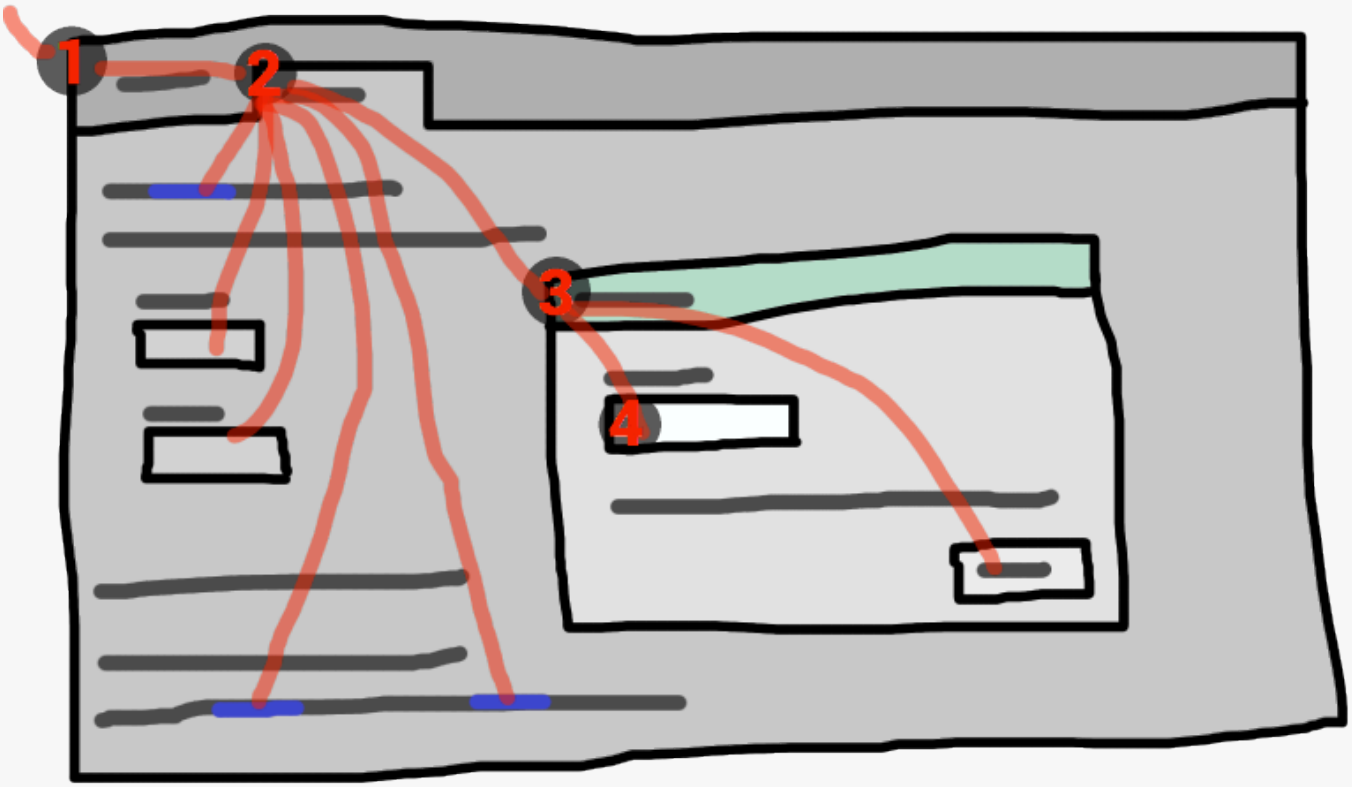
An HTML user interface typically consists of multiple interactive widgets, such as form controls, scrollable regions, links, dialog boxes, browser tabs, and so forth. These widgets form a hierarchy, with some (e.g. browser tabs, dialog boxes) containing others (e.g. links, form controls).

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[p875].

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[p875] 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[p875] element.

6.6.2 Data model §[p874]

A top-level traversable[p1037] has **system focus** when it can receive keyboard input channeled from the operating system, possibly targeted at one of its active document[p1036]'s descendant navigables[p1041].

A top-level traversable[p1037] has **user attention** when its system visibility state[p1037] is "`visible`", and it either has system focus[p874] or user agent widgets directly related to it can receive keyboard input channeled from the operating system.

Note

User attention is lost when a browser window loses focus, whereas system focus might also be lost to other system widgets in the browser window such as a location bar.

A `Document`[p135] *d* is a **fully active descendant of a top-level traversable with user attention** when *d* is fully active[p1051] and *d*'s node navigable[p1036]'s top-level traversable[p1037] has user attention[p874].

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[p874] has a **DOM anchor**, which is a `Node` object that represents the position of the focusable area[p874] in the DOM. (When the focusable area[p874] is itself a `Node`, it is its own DOM anchor[p874].) The DOM anchor[p874] is used in some APIs as a substitute for the focusable area[p874] when there is no other DOM object to represent the focusable area[p874].

The following table describes what objects can be focusable areas[p874]. The cells in the left column describe objects that can be focusable areas[p874]; the cells in the right column describe the DOM anchors[p874] for those elements. (The cells that span both columns are non-normative examples.)

Focusable area[p874]	**DOM anchor**[p874]
Examples	
Elements that meet all the following criteria: • **the element's tabindex value**[p878] **is non-null, or the element is determined by the user agent to be focusable;** • **the element is either not a shadow host, or has a shadow root whose delegates focus is false;** • **the element is not actually disabled**[p819]**;** • **the element is not inert**[p866]**;** • **the element is either being rendered**[p1488]**, delegating its rendering to its children**[p1488]**, or being used as relevant canvas fallback content**[p715]**.**	The element itself.

Example

`iframe`[p404], `dialog`[p675], `<input type=text>`[p546], sometimes `<a href="">`[p267] (depending on platform conventions).

The shapes of `area`[p489] **elements in an image map**[p491] **associated with an `img`**[p359] **element that is being rendered**[p1488] **and is not inert**[p866]**.**	The `img`[p359] element.

Example

In the following example, the `area`[p489] element creates two shapes, one on each image. The DOM anchor[p874] of the first shape is the first `img`[p359] element, and the DOM anchor[p874] of the second shape is the second `img`[p359] element.

```
<map id=wallmap><area alt="Enter Door" coords="10,10,100,200" href="door.html"></map>
...
<img src="images/innerwall.jpeg" alt="There is a white wall here, with a door." usemap="#wallmap">
...
<img src="images/outerwall.jpeg" alt="There is a red wall here, with a door." usemap="#wallmap">
```

The user-agent provided subwidgets of elements that are being rendered[p1488] **and are not actually disabled**[p819] **or inert**[p866]**.**	The element for which the focusable area[p874] is a subwidget.

Focusable area[p874]	DOM anchor[p874]
Examples	

Example

The controls in the user interface[p481] for a video[p420] element, the up and down buttons in a spin-control version of <input type=number>[p556], the part of a details[p666] element's rendering that enables the element to be opened or closed using keyboard input.

The scrollable regions of elements that are being rendered[p1488] and are not inert[p866].	The element for which the box that the scrollable region scrolls was created.

Example

The CSS 'overflow' property's 'scroll' value typically creates a scrollable region.

The viewport of a Document[p135] that has a non-null browsing context[p1046] and is not inert[p866].	The Document[p135] for which the viewport was created.

Example

The contents of an iframe[p404].

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[p876], so that a user can more easily navigate lists.

Example

Similarly, a user agent could make all elements with title[p164] attributes sequentially focusable[p876], so that their advisory information can be accessed.

¶[p875] Note

A navigable container[p1038] (e.g. an iframe[p404]) is a focusable area[p874], but key events routed to a navigable container[p1038] get immediately routed to its content navigable[p1038]'s active document[p1036]. Similarly, in sequential focus navigation a navigable container[p1038] essentially acts merely as a placeholder for its content navigable[p1038]'s active document[p1036].

One focusable area[p874] in each Document[p135] is designated the **focused area of the document**. Which control is so designated changes over time, based on algorithms in this specification.

Note

Even if a document is not fully active[p1051] and not shown to the user, it can still have a focused area of the document[p875]. If a document's fully active[p1051] state changes, its focused area of the document[p875] will stay the same.

The **currently focused area of a top-level traversable** *traversable* is the focusable area[p874]-or-null returned by this algorithm:

1. If *traversable* does not have system focus[p874], then return null.
2. Let *candidate* be *traversable*'s active document[p1036].
3. While *candidate*'s focused area[p875] is a navigable container[p1038] with a non-null content navigable[p1038]: set *candidate* to the active document[p1036] of that navigable container[p1038]'s content navigable[p1038].
4. If *candidate*'s focused area[p875] is non-null, set *candidate* to *candidate*'s focused area[p875].
5. Return *candidate*.

The **current focus chain of a top-level traversable** *traversable* is the focus chain[p876] of the currently focused area[p875] of *traversable*, if *traversable* is non-null, or an empty list otherwise.

An element that is the DOM anchor[p874] of a focusable area[p874] is said to **gain focus** when that focusable area[p874] becomes the currently focused area of a top-level traversable[p875]. When an element is the DOM anchor[p874] of a focusable area[p874] of the currently focused area of a top-level traversable[p875], it is **focused**.

The **focus chain** of a focusable area[p874] *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*.
    2. If *currentObject* is an `area`[p489] element's shape, then append that `area`[p489] element to *output*.

       Otherwise, if *currentObject*'s DOM anchor[p874] is an element that is not *currentObject* itself, then append *currentObject*'s DOM anchor[p874] to *output*.
    3. If *currentObject* is a focusable area[p874], then set *currentObject* to *currentObject*'s DOM anchor[p874]'s node document.

       Otherwise, if *currentObject* is a `Document`[p135] whose node navigable[p1036]'s parent[p1035] is non-null, then set *currentObject* to *currentObject*'s node navigable[p1036]'s parent[p1035].

       Otherwise, break.
4. Return *output*.

Note

The chain starts with subject *and (if* subject *is or can be the currently focused area of a top-level traversable[p875]) continues up the focus hierarchy up to the `Document`[p135] of the top-level traversable[p1037].*

All elements that are focusable areas[p874] are said to be **focusable**.

There are two special types of focusability for focusable areas[p874]:

- A focusable area[p874] is said to be **sequentially focusable** if it is included in its `Document`[p135]'s sequential focus navigation order[p884] and the user agent determines that it is sequentially focusable.
- A focusable area[p874] 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[p878] to be click focusable.

Note

Elements which are not focusable[p876] are not focusable areas[p874], and thus not sequentially focusable[p876] and not click focusable[p876].

Note

Being focusable[p876] is a statement about whether an element can be focused programmatically, e.g. via the `focus()`[p887] method or `autofocus`[p887] attribute. In contrast, sequentially focusable[p876] and click focusable[p876] govern how the user agent responds to user interaction: respectively, to sequential focus navigation[p884] and as activation behavior.

The user agent might determine that an element is not sequentially focusable[p876] even if it is focusable[p876] and is included in its `Document`[p135]'s sequential focus navigation order[p884], according to user preferences. For example, macOS users can set the user agent to skip non-form control elements, or can skip links when doing sequential focus navigation[p884] with just the Tab key (as opposed to using both the `Option` and Tab keys).

Similarly, the user agent might determine that an element is not click focusable[p876] even if it is focusable[p876]. For example, in some user agents, clicking on a non-editable form control does not focus it, i.e. the user agent has determined that such controls are not click focusable.

Thus, an element can be focusable[p876], but neither sequentially focusable[p876] nor click focusable[p876]. For example, in some user agents, a non-editable form-control with a negative-integer tabindex value[p878] would not be focusable via user interaction, only via programmatic APIs.

When a user activates[p871] a click focusable[p876] focusable area[p874], the user agent must run the focusing steps[p881] on the focusable area[p874] with *focus trigger* set to "`click`".

Note

Note that focusing is not an activation behavior, i.e. calling the `click()` method on an element or dispatching a synthetic `click` event on it won't cause the element to get focused.

A node is a **focus navigation scope owner** if it is a `Document`, a shadow host, a slot, or an element which is the popover trigger of an element in the popover showing state.

Each focus navigation scope owner has a **focus navigation scope**, which is a list of elements. Its contents are determined as follows:

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. If *element* is in the popover showing state and has a popover trigger set, then return *element*'s popover trigger.
6. Return *element*'s parent's associated focus navigation owner.

Then, the contents of a given focus navigation scope owner *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 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*.

6.6.3 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 sequentially focusable.

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 value** 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 value is null.

The tabindex value of a focusable area is the tabindex value of its DOM anchor.

The tabindex value 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:

- a elements that have an href attribute
- button elements

- input[p539] elements whose type[p541] attribute are not in the Hidden[p546] state
- select[p590] elements
- textarea[p606] elements
- summary[p672] elements that are the first summary[p672] element child of a details[p666] element
- Elements with a draggable[p924] 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[p894]
- Navigable containers[p1038]

If the value is a negative integer

The user agent must consider the element as a focusable area[p874], but should omit the element from any tabindex-ordered focus navigation scope[p877].

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[p878], 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[p874] and should allow the element and any focusable areas[p874] that have the element as their DOM anchor[p874] to be sequentially focusable[p876].

The relative ordering within a tabindex-ordered focus navigation scope[p877] for elements and focusable areas[p874] that belong to the same focus navigation scope[p877] and whose tabindex value[p878] 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[p874] and should allow the element and any focusable areas[p874] that have the element as their DOM anchor[p874] to be sequentially focusable[p876], and should place the element — referenced as *candidate* below — and the aforementioned focusable areas[p874] in the tabindex-ordered focus navigation scope[p877] where the element is a part of so that, relative to other elements and focusable areas[p874] that belong to the same focus navigation scope[p877], they are:

- before any focusable area[p874] whose DOM anchor[p874] is an element whose tabindex[p878] attribute has been omitted or whose value, when parsed, returns an error,
- before any focusable area[p874] whose DOM anchor[p874] is an element whose tabindex[p878] attribute has a value less than or equal to zero,
- after any focusable area[p874] whose DOM anchor[p874] is an element whose tabindex[p878] attribute has a value greater than zero but less than the value of the tabindex[p878] attribute on *candidate*,
- after any focusable area[p874] whose DOM anchor[p874] is an element whose tabindex[p878] attribute has a value equal to the value of the tabindex[p878] attribute on *candidate* but that is located earlier than *candidate* in shadow-including tree order,
- before any focusable area[p874] whose DOM anchor[p874] is an element whose tabindex[p878] attribute has a value equal to the value of the tabindex[p878] attribute on *candidate* but that is located later than *candidate* in shadow-including tree order, and
- before any focusable area[p874] whose DOM anchor[p874] is an element whose tabindex[p878] attribute has a value greater than the value of the tabindex[p878] attribute on *candidate*.

The **tabIndex** getter steps are:

1. Let *attribute* be this's tabindex[p878] attribute.
2. If *attribute* is not null:
    1. Let *parsedValue* be the result of integer parsing[p80] *attribute*'s value.

2. If *parsedValue* is not an error and is within the `long` range, then return *parsedValue*.

3. Return 0 if this is an `a`, `area`, `button`, `frame`, `iframe`, `input`, `object`, `select`, `textarea`, or SVG `a` element, or MathML `a` element, or is a `summary` element that is a summary for its parent details; otherwise −1.

Note

The varying default value based on element type is a historical artifact.

6.6.4 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 navigable, given an optional string *focus trigger* (default "`other`"), 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 belongs.

↪ **If *focus target* is an element with one or more scrollable regions that are focusable areas**

Return the element's first scrollable region, according to a pre-order, depth-first traversal of the flat tree. [CSSSCOPING]

↪ **If *focus target* is the document element of its `Document`**

Return the `Document`'s viewport.

↪ **If *focus target* is a navigable**

Return the navigable's active document.

↪ **If *focus target* is a navigable container with a non-null content navigable**

Return the navigable container's content navigable's active document.

↪ **If *focus target* is a shadow host whose shadow root's delegates focus is true**

1. Let *focusedElement* be the currently focused area of a top-level traversable's DOM anchor.
2. If *focus target* is a shadow-including inclusive ancestor of *focusedElement*, then return *focusedElement*.
3. Return the focus delegate for *focus target* given *focus trigger*.

Note

For sequential focusability, the handling of shadow hosts and delegates focus is done when constructing the sequential focus navigation order. That is, the focusing steps will never be called on such shadow hosts as part of sequential focus navigation.

↪ **Otherwise**

Return null.

The **focus delegate** for a *focusTarget*, given an optional string *focusTrigger* (default "`other`"), is given by the following steps:

1. If *focusTarget* is a shadow host and its shadow root's delegates focus is false, then return null.
2. Let *whereToLook* be *focusTarget*.
3. If *whereToLook* is a shadow host, then set *whereToLook* to *whereToLook*'s shadow root.
4. Let *autofocusDelegate* be the autofocus delegate for *whereToLook* given *focusTrigger*.
5. If *autofocusDelegate* is not null, then return *autofocusDelegate*.
6. For each *descendant* of *whereToLook*'s descendants, in tree order:
    1. Let *focusableArea* be null.
    2. If *focusTarget* is a `dialog` element and *descendant* is sequentially focusable, then set *focusableArea* to

descendant.

3. Otherwise, if *focusTarget* is not a `dialog`[p675] and *descendant* is a focusable area[p874], set *focusableArea* to *descendant*.

4. Otherwise, set *focusableArea* to the result of getting the focusable area[p880] for *descendant* given *focusTrigger*.

   Note

   This step can end up recursing, i.e., the get the focusable area[p880] steps might return the focus delegate[p880] of descendant*.*

5. If *focusableArea* is not null, then return *focusableArea*.

Note

It's important that we are not *looking at the shadow-including descendants here, but instead only at the descendants. Shadow hosts are instead handled by the recursive case mentioned above.*

7. Return null.

Note

The above algorithm essentially returns the first suitable focusable area[p874] where the path between its DOM anchor[p874] and focusTarget *delegates focus at any shadow tree boundaries.*

The **autofocus delegate** for a *focus target* given a *focus trigger* is given by the following steps:

1. For each descendant *descendant* of *focus target*, in tree order:

   1. If *descendant* does not have an `autofocus`[p887] content attribute, then continue.

   2. Let *focusable area* be *descendant*, if *descendant* is a focusable area[p874]; otherwise let *focusable area* be the result of getting the focusable area[p880] for *descendant* given *focus trigger*.

   3. If *focusable area* is null, then continue.

   4. If *focusable area* is not click focusable[p876] and *focus trigger* is "`click`", then continue.

   5. Return *focusable area*.

2. Return null.

The **focusing steps** for an object *new focus target* that is either a focusable area[p874], or an element that is not a focusable area[p874], or a navigable[p1035], 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[p874], then set *new focus target* to the result of getting the focusable area[p880] for *new focus target*, given *focus trigger* if it was passed.

2. If *new focus target* is null:

   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 navigable container[p1038] with non-null content navigable[p1038], then set *new focus target* to the content navigable[p1038]'s active document[p1036].

4. If *new focus target* is a focusable area[p874] and its DOM anchor[p874] is inert[p866], then return.

5. If *new focus target* is the currently focused area of a top-level traversable[p875], then return.

6. Let *old chain* be the current focus chain of the top-level traversable[p875] in which *new focus target* finds itself.

7. Let *new chain* be the focus chain[p876] of *new focus target*.

8. Run the focus update steps[p882] with *old chain*, *new chain*, and *new focus target* respectively.

User agents must immediately[p44] run the focusing steps[p881] for a focusable area[p874] or navigable[p1035] *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[p874] or an element that is not a focusable area[p874] are as follows:

1. If *old focus target* is a shadow host whose shadow root's delegates focus is true, and *old focus target*'s shadow root is a shadow-including inclusive ancestor of the currently focused area of a top-level traversable[p875]'s DOM anchor[p874], then set *old focus target* to that currently focused area of a top-level traversable[p875].

2. If *old focus target* is inert[p866], then return.

3. If *old focus target* is an `area`[p489] element and one of its shapes is the currently focused area of a top-level traversable[p875], 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 traversable[p875], then let *old focus target* be that currently focused area of a top-level traversable[p875].

4. Let *old chain* be the current focus chain of the top-level traversable[p875] in which *old focus target* finds itself.

5. If *old focus target* is not one of the entries in *old chain*, then return.

6. If *old focus target* is not a focusable area[p874], then return.

7. Let *topDocument* be *old chain*'s last entry.

8. If *topDocument*'s node navigable[p1036] has system focus[p874], then run the focusing steps[p881] for *topDocument*'s viewport.

   Otherwise, apply any relevant platform-specific conventions for removing system focus[p874] from *topDocument*'s node navigable[p1036], and run the focus update steps[p882] given *old chain*, an empty list, and null.

Note

The unfocusing steps[p882] do not always result in the focus changing, even when applied to the currently focused area of a top-level traversable[p875]. For example, if the currently focused area of a top-level traversable[p875] is a viewport, then it will usually keep its focus regardless until another focusable area[p874] is explicitly focused with the focusing steps[p881].

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 *entry* in *old chain*, in order, run these substeps:

   1. If *entry* is an `input`[p539] element, and the `change`[p1575] event applies[p542] to the element, and the element does not have a defined activation behavior, and the user has changed the element's value[p626] or its list of selected files[p563] while the control was focused without committing that change (such that it is different to what it was when the control was first focused):

      1. Set *entry*'s user validity[p626] to true.

      2. Fire an event named `change`[p1575] 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`[p135] object, let *blur event target* be that `Document`[p135] object's relevant global object[p1151].

      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[p883] named `blur`[p1574] at *blur event target*, with *related blur target* as the related target.

   ¶[p882] Note

   In some cases, e.g., if entry *is an* `area`[p489] *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[p874], and the focused area of the document[p875] is not *entry*:

    1. Set *document*'s relevant global object[p1151]'s navigation API[p995]'s focus changed during ongoing navigation[p1007] to true.

    2. Designate *entry* as the focused area of the document[p875].

2. If *entry* is an element, let *focus event target* be *entry*.

    If *entry* is a `Document`[p135] object, let *focus event target* be that `Document`[p135] object's relevant global object[p1151].

    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[p883] named `focus`[p1575] at *focus event target*, with *related focus target* as the related target.

> **Note**
>
> *In some cases, e.g. if* entry *is an* `area`[p489] *element's shape, a scrollable region, or a viewport, no event is fired.*

To **fire a focus event** named *e* at an element or a `Window`[p965] object *target* given a related target *relatedTarget*:

1. Let *view* be *target* if *target* is a `Window`[p965] object; otherwise *target*'s node document's relevant global object[p1151].

2. Fire an event named *e* at *target*, using `FocusEvent`, with the `relatedTarget` attribute initialized to *relatedTarget*, the `view` attribute initialized to *view*, and the composed flag set.

When a key event is to be routed in a top-level traversable[p1037], the user agent must run the following steps:

1. Let *target area* be the currently focused area of the top-level traversable[p875].

2. Assert: *target area* is not null, since key events are only routed to top-level traversables[p1037] that have system focus[p874]. Therefore, *target area* is a focusable area[p874].

3. Let *target node* be *target area*'s DOM anchor[p874].

4. If *target node* is a `Document`[p135] that has a body element[p144], then let *target node* be the body element[p144] of that `Document`[p135].

    Otherwise, if *target node* is a `Document`[p135] object that has a non-null document element, then let *target node* be that document element.

5. If *target node* is not inert[p866]:

    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[p1218] at *target node*.

The **has focus steps**, given a `Document`[p135] object *target*, are as follows:

1. If *target*'s node navigable[p1036]'s top-level traversable[p1037] does not have system focus[p874], then return false.

2. Let *candidate* be *target*'s node navigable[p1036]'s top-level traversable[p1037]'s active document[p1046].

3. While true:

    1. If *candidate* is *target*, then return true.

    2. If the focused area[p875] of *candidate* is a navigable container[p1038] with a non-null content navigable[p1038], then set *candidate* to the active document[p1036] of that navigable container[p1038]'s content navigable[p1038].

    3. Otherwise, return false.

6.6.5 Sequential focus navigation §[p884]

Each Document[p135] has a **sequential focus navigation order**, which orders some or all of the focusable areas[p874] in the Document[p135] relative to each other. Its contents and ordering are given by the flattened tabindex-ordered focus navigation scope[p877] of the Document[p135].

Note

Per the rules defining the flattened tabindex-ordered focus navigation scope[p877], the ordering is not necessarily related to the tree order of the Document[p135].

If a focusable area[p874] is omitted from the sequential focus navigation order[p884] of its Document[p135], then it is unreachable via sequential focus navigation[p884].

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[p884] to the target element[p1104] when navigating to a fragment[p1071].

A **sequential focus direction** is one of two possible values: "**forward**", or "**backward**". They are used in the below algorithms to describe the direction in which sequential focus travels at the user's request.

A **selection mechanism** is one of two possible values: "**DOM**", or "**sequential**". They are used to describe how the sequential navigation search algorithm[p885] finds the focusable area[p874] it returns.

When the user requests that focus move from the currently focused area of a top-level traversable[p875] to the next or previous focusable area[p874] (e.g., as the default action of pressing the tab key), or when the user requests that focus sequentially move to a top-level traversable[p1037] 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 traversable[p875], if the user requested to move focus sequentially from there, or else the top-level traversable[p1037] itself, if the user instead requested to move focus from outside the top-level traversable[p1037].
2. If there is a sequential focus navigation starting point[p884] defined and it is inside *starting point*, then let *starting point* be the sequential focus navigation starting point[p884] instead.
3. Let *direction* be "forward[p884]" if the user requested the next control, and "backward[p884]" 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[p884]" if *starting point* is a navigable[p1035] or if *starting point* is in its Document[p135]'s sequential focus navigation order[p884].

   Otherwise, *starting point* is not in its Document[p135]'s sequential focus navigation order[p884]; let *selection mechanism* be "DOM[p884]".
5. Let *candidate* be the result of running the sequential navigation search algorithm[p885] with *starting point*, *direction*, and *selection mechanism*.
6. If *candidate* is not null, then run the focusing steps[p881] for *candidate* and return.
7. Otherwise, unset the sequential focus navigation starting point[p884].
8. If *starting point* is a top-level traversable[p1037], or a focusable area[p874] in the top-level traversable[p1037], 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[p876] control before the browser's rendering area would be the control to focus.

If the user agent has no sequentially focusable[p876] controls — a kiosk-mode browser, for instance — then the user agent may instead restart these steps with the *starting point* being the top-level traversable[p1037] itself.

9. Otherwise, *starting point* is a focusable area[p874] in a child navigable[p1039]. Set *starting point* to that child navigable[p1039]'s parent[p1035] and return to the step labeled *loop*.

The **sequential navigation search algorithm**, given a focusable area[p874] *starting point*, sequential focus direction[p884] *direction*, and selection mechanism[p884] *selection mechanism*, consists of the following steps. They return a focusable area[p874]-or-null.

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*.

	***direction* is "forward[p884]"**	***direction* is "backward[p884]"**
***starting point* is a navigable[p1035]**	Let *candidate* be the first suitable sequentially focusable area[p885] in *starting point*'s active document[p1036], if any; or else null.	Let *candidate* be the last suitable sequentially focusable area[p885] in *starting point*'s active document[p1036], if any; or else null.
***selection mechanism* is "DOM[p884]"**	Let *candidate* be the suitable sequentially focusable area[p885], that appears nearest after *starting point* in *starting point*'s Document[p135], in shadow-including tree order, if any; or else null. Note *In this case,* starting point *does not necessarily belong to its* Document[p135]*'s sequential focus navigation order[p884], so we'll select the suitable[p885] item from that list that comes* after starting point *in shadow-including tree order.*	Let *candidate* be the suitable sequentially focusable area[p885], that appears nearest before *starting point* in *starting point*'s Document[p135], in shadow-including tree order, if any; or else null.
***selection mechanism* is "sequential[p884]"**	Let *candidate* be the first suitable sequentially focusable area[p885] after *starting point*, in *starting point*'s Document[p135]'s sequential focus navigation order[p884], if any; or else null.	Let *candidate* be the last suitable sequentially focusable area[p885] before *starting point*, in *starting point*'s Document[p135]'s sequential focus navigation order[p884], if any; or else null.

   A **suitable sequentially focusable area** is a focusable area[p874] whose DOM anchor[p874] is not inert[p866] and is sequentially focusable[p876].

2. If *candidate* is a navigable container[p1038] with a non-null content navigable[p1038]:

   1. Let *recursive candidate* be the result of running the sequential navigation search algorithm[p885] with *candidate*'s content navigable[p1038], *direction*, and "sequential[p884]".

   2. If *recursive candidate* is null, then return the result of running the sequential navigation search algorithm[p885] with *candidate*, *direction*, and *selection mechanism*.

   3. Otherwise, set *candidate* to *recursive candidate*.

3. Return *candidate*.

6.6.7 Focus management APIs §[p885]

IDL

```
dictionary FocusOptions {
  boolean preventScroll = false;
  boolean focusVisible;
};
```

For web developers (non-normative)

***documentOrShadowRoot*.activeElement**[p886]

Returns the deepest element in *documentOrShadowRoot* through which or to which key events are being routed. This is, roughly speaking, the focused element in the document.

For the purposes of this API, when a child navigable[p1039] is focused, its container[p1038] is focused[p875] within its parent[p1035]'s active document[p1036]. For example, if the user moves the focus to a text control in an `iframe`[p404], the `iframe`[p404] is the element returned by the `activeElement`[p886] API in the `iframe`[p404]'s node document.

Similarly, when the focused element is in a different node tree than *documentOrShadowRoot*, the element returned will be the host that's located in the same node tree as *documentOrShadowRoot* if *documentOrShadowRoot* is a shadow-including inclusive ancestor of the focused element, and null if not.

***document*.hasFocus**[p886]**()**

Returns true if key events are being routed through or to *document*; otherwise, returns false. Roughly speaking, this corresponds to *document*, or a document nested inside *document*, being focused.

***window*.focus**[p886]**()**

Moves the focus to *window*'s navigable[p966], if any.

***element*.focus**[p887]**()**

***element*.focus**[p887]**({ preventScroll**[p887]**, focusVisible**[p887]** })**

Moves the focus to *element*.

If *element* is a navigable container[p1038], moves the focus to its content navigable[p1038] instead.

By default, this method also scrolls *element* into view. Providing the `preventScroll`[p887] option and setting it to true prevents this behavior.

By default, user agents use implementation-defined heuristics to determine whether to indicate focus via a focus ring. Providing the `focusVisible`[p887] option and setting it to true will ensure the focus ring is always visible.

***element*.blur**[p887]**()**

Moves the focus to the viewport. Use of this method is discouraged; if you want to focus the viewport, call the `focus()`[p887] method on the `Document`[p135]'s document element.

Do not use this method to hide the focus ring if you find the focus ring unsightly. Instead, use the `:focus-visible` pseudo-class to override the 'outline' property, and provide a different way to show what element is focused. Be aware that if an alternative focusing style isn't made available, the page will be significantly less usable for people who primarily navigate pages using a keyboard, or those with reduced vision who use focus outlines to help them navigate the page.

Example

For example, to hide the outline from `textarea`[p606] elements and instead use a yellow background to indicate focus, you could use:

```css
textarea:focus-visible { outline: none; background: yellow; color: black; }
```

The `DocumentOrShadowRoot`[p137] **`activeElement`** getter steps are:

1. Let *candidate* be this's node document's focused area[p875]'s DOM anchor[p874].
2. Set *candidate* to the result of retargeting *candidate* against this.
3. If *candidate*'s root is not this, then return null.
4. If *candidate* is not a `Document`[p135] object, then return *candidate*.
5. If *candidate* has a body element[p144], then return that body element[p144].
6. If *candidate*'s document element is non-null, then return that document element.
7. Return null.

The `Document`[p135] **`hasFocus()`** method steps are to return the result of running the has focus steps[p883] given this.

The `Window`[p965] **`focus()`** method steps are:

1. Let *current* be this's navigable[p966].
2. If *current* is null, then return.
3. If the allow focus steps[p887] given *current*'s active document[p1036] return false, then return.
4. Run the focusing steps[p881] with *current*.
5. If *current* is a top-level traversable[p1037], user agents are encouraged to trigger some sort of notification to indicate to the user that the page is attempting to gain focus.

The `Window`[p965] **`blur()`** method steps are to do nothing.

✔ MDN

Note

Historically, the `focus()`[p886] and `blur()`[p887] methods actually affected the system-level focus of the system widget (e.g., tab or window) that contained the navigable[p1035], but hostile sites widely abuse this behavior to the user's detriment.

The `HTMLOrSVGOrMathMLElement`[p150] **`focus(options)`** method steps are:

1. If the allow focus steps[p887] given this's node document return false, then return.
2. Run the focusing steps[p881] for this.
3. If *options*["**`focusVisible`**"] is true, or does not exist but in an implementation-defined way the user agent determines it would be best to do so, then indicate focus.
4. If *options*["**`preventScroll`**"] is false, then scroll a target into view given this, "`auto`", "`center`", and "`center`".

The `HTMLOrSVGOrMathMLElement`[p150] **`blur()`** method steps are:

1. The user agent should run the unfocusing steps[p882] given this.

   User agents may instead selectively or uniformly do nothing, for usability reasons.

Example

For example, if the `blur()`[p887] 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.

The **allow focus steps**, given a `Document`[p135] object *target*, are:

1. If *target* is allowed to use[p412] the "`focus-without-user-activation`[p79]" feature, then return true.
2. If *target*'s relevant global object[p1151] has transient activation[p868], then return true.
3. Return false.

6.6.7 The `autofocus`[p887] attribute §[p887]

The **`autofocus`** content attribute allows the author to indicate that an element is to be focused as soon as the page is loaded, allowing the user to just start typing without having to manually focus the main element.

When the `autofocus`[p887] attribute is specified on an element inside `dialog`[p675] elements or HTML elements[p46] whose `popover`[p925] attribute is set, then it will be focused when the dialog or popover becomes shown.

The `autofocus`[p887] attribute is a boolean attribute[p79].

To find the **nearest ancestor autofocus scoping root element** given an `Element` *element*:

1. If *element* is a `dialog`[p675] element, then return *element*.
2. If *element*'s `popover`[p925] attribute is not in the No Popover[p926] state, then return *element*.

3. Let *ancestor* be *element*.
4. While *ancestor* has a parent element:
    1. Set *ancestor* to *ancestor*'s parent element.
    2. If *ancestor* is a `dialog` element, then return *ancestor*.
    3. If *ancestor*'s `popover` attribute is not in the No Popover state, then return *ancestor*.
5. Return *ancestor*.

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* is not fully active, then return.
4. If *target*'s active sandboxing flag set has the sandboxed automatic features browsing context flag, then return.
5. If the allow focus steps given *target* return false, then return.
6. Let *topDocument* be *target*'s node navigable's top-level traversable's active document.
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.

Note

We do not check if an element is a focusable area before storing it in the autofocus candidates list, because even if it is not a focusable area when it is inserted, it could become one by the time flush autofocus candidates sees it.

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* has non-null target element:
    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 node navigable's top-level traversable is not the same as *topDocument*'s node navigable, then remove *element* from *candidates*, and continue.
    5. If *doc*'s script-blocking style sheet set is not empty, then return.

Note

In this case, element *is the currently-best candidate, but* doc *is not ready for autofocusing. We'll try again next time flush autofocus candidates[p888] is called.*

6. Remove *element* from *candidates*.
7. Let *inclusiveAncestorDocuments* be a list consisting of the active document[p1036] of *doc*'s inclusive ancestor navigables[p1041].
8. If any Document[p135] in *inclusiveAncestorDocuments* has non-null target element[p1104], then continue.
9. Let *target* be *element*.
10. If *target* is not a focusable area[p874], then set *target* to the result of getting the focusable area[p880] for *target*.

Note

Autofocus candidates[p888] can contain elements which are not focusable areas[p874]. In addition to the special cases handled in the get the focusable area[p880] algorithm, this can happen because a non-focusable area[p874] element with an autofocus[p887] attribute was inserted into a document[p47] and it never became focusable, or because the element was focusable but its status changed while it was stored in autofocus candidates[p888].

11. If *target* is not null:
    1. Empty *candidates*.
    2. Set *topDocument*'s autofocus processed flag[p888] to true.
    3. Run the focusing steps[p881] for *target*.

Note

This handles the automatic focusing during document load. The show()[p678] and showModal()[p679] methods of dialog[p675] elements also processes the autofocus[p887] attribute.

Note

Focusing the element does not imply that the user agent has to focus the browser window if it has lost focus.

Example

In the following snippet, the text control would be focused when the document was loaded.

```
<input maxlength="256" name="q" value="" autofocus>
<input type="submit" value="Search">
```

Example

The autofocus[p887] attribute applies to all elements, not just to form controls. This allows examples such as the following:

```
<div contenteditable autofocus>Edit <strong>me!</strong><div>
```

6.7 Assigning keyboard shortcuts §[p889]

6.7.1 Introduction §[p889]

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[p890] 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[p890]

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[p890] 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[p892] 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 possible key.

```
<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 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 "⌥⎋", and an Emacs browser might use "M-ESC" or "ESC ESC".

In general, therefore, it is unwise to attempt to parse the value returned from the accessKeyLabel[p892] IDL attribute.

6.7.2 The accesskey attribute §[p890]

All HTML elements[p46] may have the accesskey[p890] content attribute set. The accesskey[p890] 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[p99] 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:

```
<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`:

```
<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.

```
<input type=submit accesskey="N @ 1" value="Compose">
...
<script>
 function labelButton(button) {
   if (button.accessKeyLabel)
     button.value += ' (' + button.accessKeyLabel + ')';
 }
 var inputs = document.getElementsByTagName('input');
 for (var i = 0; i < inputs.length; i += 1) {
   if (inputs[i].type == "submit")
     labelButton(inputs[i]);
 }
</script>
```

On one user agent, the button's label might become "`Compose (⌘N)`". On another, it might become "`Compose (Alt+⇧+1)`". If the user agent doesn't assign a key, it will be just "`Compose`". The exact string depends on what the assigned access key[p891] is, and on how the user agent represents that key combination.

6.7.3 Processing model §[p891]

An element's **assigned access key** is a key combination derived from the element's accesskey[p890] content attribute. Initially, an element must not have an assigned access key[p891].

Whenever an element's accesskey[p890] attribute is set, changed, or removed, the user agent must update the element's assigned access key[p891] by running the following steps:

1. If the element has no accesskey[p890] 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[p891] and return.

4. *Fallback*: Optionally, the user agent may assign a key combination of its choosing as the element's assigned access key[p891] and then return.
5. If this step is reached, the element has no assigned access key[p891].

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[p891] unless the accesskey[p890] content attribute is changed or the element is moved to another Document[p135].

When the user presses the key combination corresponding to the assigned access key[p891] for an element, if the element defines a command[p672], the command's Hidden State[p673] facet is false (visible), the command's Disabled State[p673] facet is also false (enabled), the element is in a document that has a non-null browsing context[p1046], and neither the element nor any of its ancestors has a hidden[p862] attribute specified, then the user agent must trigger the Action[p673] of the command.

Note

User agents might expose[p673] elements that have an accesskey[p890] attribute in other ways as well, e.g. in a menu displayed in response to a specific key combination.

The **accessKeyLabel** IDL attribute must return a string that represents the element's assigned access key[p891], if any. If the element does not have one, then the IDL attribute must return the empty string.

6.8 Editing §[p89 2]

6.8.1 Making document regions editable: The contenteditable[p892] content attribute §[p89 2]

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[p80] with the following keywords and states:

Keyword	State	Brief description
true	**True**	The element is editable.
false	**False**	The element is not editable.
plaintext-only	**Plaintext-Only**	Only the element's raw text content is editable; rich formatting is disabled.

The attribute's *missing value default*[p80] and *invalid value default*[p80] are both the **Inherit** state. The inherit state indicates that the element is editable (or not) based on the parent element's state. The attribute's *empty value default*[p80] is the True[p892] state.

Example

For example, consider a page that has a form[p532] and a textarea[p606] to publish a new article, where the user is expected to write the article using HTML:

```
<form method=POST>
 <fieldset>
  <legend>New article</legend>
  <textarea name=article>&lt;p>Hello world.&lt;/p></textarea>
 </fieldset>
 <p><button>Publish</button></p>
</form>
```

When scripting is enabled, the textarea[p606] element could be replaced with a rich text control instead, using the contenteditable[p892] attribute:

```
<form method=POST>
 <fieldset>
  <legend>New article</legend>
  <textarea id=textarea name=article>&lt;p>Hello world.&lt;/p></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. [EXECCOMMAND] [SELECTION] [DOM]

Example

The `contenteditable` attribute can also be used to great effect:

```
<!doctype html>
<html lang=en>
<title>Live CSS editing!</title>
<style style=white-space:pre contenteditable>
html { margin:.2em; font-size:2em; color:lime; background:purple }
head, title, style { display:block }
body { display:none }
</style>
```

For web developers (non-normative)

***element*.contentEditable [= *value*]**

Returns "true", "plaintext-only", "false", or "inherit", based on the state of the `contenteditable` attribute.

Can be set, to change that state.

Throws a "SyntaxError" DOMException if the new value isn't one of those strings.

***element*.isContentEditable**

Returns true if the element is editable; otherwise, returns false.

The **contentEditable** IDL attribute, on getting, must return the string "true" if the content attribute is set to the True state, "plaintext-only" if the content attribute is set to the Plaintext-Only state, "false" if the content attribute is set to the False state, and "inherit" otherwise. On setting, if the new value is an ASCII case-insensitive match for the string "inherit", then the content attribute must be removed, if the new value is an ASCII case-insensitive match for the string "true", then the content attribute must be set to the string "true", if the new value is an ASCII case-insensitive match for the string "plaintext-only", then the content attribute must be set to the string "plaintext-only", if the new value is an ASCII case-insensitive match for the string "false", then the content attribute must be set to the string "false", and otherwise the attribute setter must throw a "SyntaxError" DOMException.

The **isContentEditable** IDL attribute, on getting, must return true if the element is either an editing host or editable, and false otherwise.

6.8.2 Making entire documents editable: the designMode getter and setter

For web developers (non-normative)

***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[p135] 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[p894] is true; otherwise "off".

The designMode[p894] 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[p894] is false:
    1. Set this's design mode enabled[p894] 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[p881] for this's document element, if non-null.
3. If *value* is "off", then set this's design mode enabled[p894] to false.

6.8.3 Best practices for in-page editors §[p894]

Authors are encouraged to set the 'white-space' property on editing hosts[p894] 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.

6.8.4 Editing APIs §[p894]

An **editing host** is either an HTML element[p46] with its contenteditable[p892] attribute in the *true* state or *plaintext-only* state, or a child HTML element[p46] of a Document[p135] whose design mode enabled[p894] is true.

The definition of the terms **active range**, **editing host of**, and **editable**, the user interface requirements of elements that are editing hosts[p894] or editable, the **execCommand()**, **queryCommandEnabled()**, **queryCommandIndeterm()**, **queryCommandState()**, **queryCommandSupported()**, and **queryCommandValue()** methods, text selections, and the **delete the selection** algorithm are defined in *execCommand*. [EXECCOMMAND][p1582]

6.8.5 Spelling and grammar checking §[p894]

User agents can support the checking of spelling and grammar of editable text, either in form controls (such as the value of textarea[p606] elements), or in elements in an editing host[p894] (e.g. using contenteditable[p892]).

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[p895] attribute.

false-by-default

The element will never be checked for spelling and grammar unless spellchecking is explicitly enabled through the spellcheck[p895] 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[p80] with the following keywords and states:

Keyword	State	Brief description
true	**True**	Spelling and grammar will be checked.
false	**False**	Spelling and grammar will not be checked.

The attribute's *missing value default*[p80] and *invalid value default*[p80] are both the **Default** state. The default state indicates that the element is to act according to a default behavior, possibly based on the parent element's own spellcheck[p895] state, as defined below. The attribute's *empty value default*[p80] is the True[p895] state.

For web developers (non-normative)

***element*.spellcheck[p895] [= *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[p895] content attribute.

The **spellcheck** IDL attribute, on getting, must return true if the element's spellcheck[p895] content attribute is in the True[p895] state, or if the element's spellcheck[p895] content attribute is in the Default[p895] state and the element's default behavior[p894] is true-by-default[p895], or if the element's spellcheck[p895] content attribute is in the Default[p895] state and the element's default behavior[p894] is inherit-by-default[p895] and the element's parent element's spellcheck[p895] IDL attribute would return true; otherwise, if none of those conditions applies, then the attribute must instead return false.

Note

The spellcheck[p895] IDL attribute is not affected by user preferences that override the spellcheck[p895] content attribute, and therefore might not reflect the actual spellchecking state.

On setting, if the new value is true, then the element's spellcheck[p895] content attribute must be set to "`true`", otherwise it must be set to "`false`".

User agents should only consider the following pieces of text as checkable for the purposes of this feature:

- The value[p626] of input[p539] elements whose type[p541] attributes are in the Text[p546], Search[p546], URL[p547], or Email[p548] states and that are mutable[p626] (i.e. that do not have the readonly[p570] attribute specified and that are not disabled[p630]).
- The value[p626] of textarea[p606] elements that do not have a readonly[p608] attribute and that are not disabled[p630].
- Text in Text nodes that are children of editing hosts[p894] 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[p539] and textarea[p606] 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`[p895] content attribute, then: if that attribute is in the True[p895] state, then checking is enabled; otherwise, if that attribute is in the False[p895] state, then checking is disabled.
4. Otherwise, if there is an ancestor element with a `spellcheck`[p895] content attribute that is not in the Default[p895] state, then: if the nearest such ancestor's `spellcheck`[p895] content attribute is in the True[p895] state, then checking is enabled; otherwise, checking is disabled.
5. Otherwise, if the element's default behavior[p894] is true-by-default[p895], then checking is enabled.
6. Otherwise, if the element's default behavior[p894] is false-by-default[p895], 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`[p539] element attributes such as `pattern`[p573] 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.

Example

The element with ID "a" in the following example would be the one used to determine if the word "Hello" is checked for spelling errors. In this example, it would not be.

```
<div contenteditable="true">
 <span spellcheck="false" id="a">Hell</span><em>o!</em>
</div>
```

The element with ID "b" in the following example would have checking enabled (the leading space character in the attribute's value on the `input`[p539] element causes the attribute to be ignored, so the ancestor's value is used instead, regardless of the default).

```
<p spellcheck="true">
 <label>Name: <input spellcheck=" false" id="b"></label>
</p>
```

Note

This specification does not define the user interface for spelling and grammar checkers. A user agent could offer on-demand checking, could perform continuous checking while the checking is enabled, or could use other interfaces.

6.8.6 Writing suggestions §[p896]

User agents offer writing suggestions as users type into editable regions, either in form controls (e.g., the `textarea`[p606] element) or in elements in an editing host[p894].

The **`writingsuggestions`** content attribute is an enumerated attribute[p80] with the following keywords and states:

Keyword	State	Brief description
`true`	**True**	Writing suggestions should be offered on this element.
`false`	**False**	Writing suggestions should not be offered on this element.

The attribute's *missing value default*[p80] is the **Default** state. The default state indicates that the element is to act according to a default behavior, possibly based on the parent element's own `writingsuggestions`[p896] state, as defined below.

The attribute's *invalid value default*[p80] and *empty value default*[p80] are both the True[p896] state.

For web developers (non-normative)

***element*.writingSuggestions**[p897] **[= *value*]**

Returns "true" if the user agent is to offer writing suggestions under the scope of the element; otherwise, returns "false".

Can be set, to override the default and set the writingsuggestions[p896] content attribute.

The **computed writing suggestions value** of a given *element* is determined by running the following steps:

1. If *element*'s writingsuggestions[p896] content attribute is in the False[p896] state, return "false".
2. If *element*'s writingsuggestions[p896] content attribute is in the Default[p896] state, *element* has a parent element, and the computed writing suggestions value[p897] of *element*'s parent element is "false", then return "false".
3. Return "true".

The **writingSuggestions** getter steps are:

1. Return this's computed writing suggestions value[p897].

Note

The writingSuggestions[p897] IDL attribute is not affected by user preferences that override the writingsuggestions[p896] content attribute, and therefore might not reflect the actual writing suggestions state.

User agents should only offer suggestions within an element's scope if the result of running the following algorithm given *element* returns true:

1. If the user has disabled writing suggestions, then return false.
2. If none of the following conditions are true:
   - *element* is an input[p539] element whose type[p541] attribute is in either the Text[p546], Search[p546], Telephone[p547], URL[p547], or Email[p548] state and *element* is mutable[p626];
   - *element* is a textarea[p606] element that is mutable[p626]; or
   - *element* is an editing host[p894] or is editable,

   then return false.
3. If *element* has an inclusive ancestor with a writingsuggestions[p896] content attribute that's not in the Default[p896] and the nearest such ancestor's writingsuggestions[p896] content attribute is in the False[p896] state, then return false.
4. Otherwise, return true.

Note

This specification does not define the user interface for writing suggestions. A user agent could offer on-demand suggestions, continuous suggestions as the user types, inline suggestions, autofill-like suggestions in a popup, or could use other interfaces.

6.8.7 Autocapitalization §[p897]

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[p898] attribute allows authors to control such behavior.

The autocapitalize[p898] 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[p898] attribute can be used on an editing host[p894] to control autocapitalization behavior for the hosted editable region, on an input[p539] or textarea[p606] element to control the behavior for inputting text into that element, or on a form[p532] element to control the default behavior for all autocapitalize-and-autocorrect inheriting elements[p532] associated with the form[p532] element.

The autocapitalize[p898] attribute never causes autocapitalization to be enabled for input[p539] elements whose type[p541] attribute is in one of the URL[p547], Email[p548], or Password[p550] states. (This behavior is included in the used autocapitalization hint[p899] 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 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.

The **autocapitalize** attribute is an enumerated attribute[p80] whose states are the possible autocapitalization hints[p898]. The autocapitalization hint[p898] specified by the attribute's state combines with other considerations to form the used autocapitalization hint[p899], which informs the behavior of the user agent. The keywords for this attribute and their state mappings are as follows:

Keyword	State
off	None[p898]
none	
on	Sentences[p898]
sentences	
words	Words[p898]
characters	Characters[p898]

The attribute's *missing value default*[p80] is the Default[p898] state, and its *invalid value default*[p80] is the Sentences[p898] state.

For web developers (non-normative)

***element*.autocapitalize[p898] [= *value*]**

Returns the current autocapitalization state for the element, or an empty string if it hasn't been set. Note that for input[p539] and textarea[p606] elements that inherit their state from a form[p532] element, this will return the autocapitalization state of the form[p532] 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[p894]).

Can be set, to set the autocapitalize[p898] 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[p898] 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-and-autocorrect inheriting element[p532] and has a non-null form owner[p626], return the own autocapitalization hint[p898] of *element*'s form owner[p626].
3. Return Default[p898].

The **autocapitalize** getter steps are to:

1. Let *state* be the own autocapitalization hint[p898] of this.

2. If *state* is Default[p898], then return the empty string.
3. If *state* is None[p898], then return "`none`[p898]".
4. If *state* is Sentences[p898], then return "`sentences`[p898]".
5. Return the keyword value corresponding to *state*.

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[p898] 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[p899] in certain circumstances.

The used autocapitalization hint[p899] for an element *element* is computed using the following algorithm:

1. If *element* is an `input`[p539] element whose `type`[p541] attribute is in one of the URL[p547], Email[p548], or Password[p550] states, then return Default[p898].
2. If *element* is an `input`[p539] element or a `textarea`[p606] element, then return *element*'s own autocapitalization hint[p898].
3. If *element* is an editing host[p894] or an editable element, then return the own autocapitalization hint[p898] 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.

6.8.8 Autocorrection §[p899]

Some methods of entering text assist users by automatically correcting misspelled words while typing, a process also known as autocorrection. User agents can support autocorrection of editable text, either in form controls (such as the value of `textarea`[p606] elements), or in elements in an editing host[p894] (e.g., using `contenteditable`[p892]). Autocorrection may be accompanied by user interfaces indicating that text is about to be autocorrected or has been autocorrected, and is commonly performed when inserting punctuation characters, spaces, or new paragraphs after misspelled words. The `autocorrect`[p899] attribute allows authors to control such behavior.

The `autocorrect`[p899] attribute can be used on an editing host to control autocorrection behavior for the hosted editable region, on an `input`[p539] or `textarea`[p606] element to control the behavior when inserting text into that element, or on a `form`[p532] element to control the default behavior for all autocapitalize-and-autocorrect inheriting elements[p532] associated with the `form`[p532] element.

The `autocorrect`[p899] attribute never causes autocorrection to be enabled for `input`[p539] elements whose `type`[p541] attribute is in one of the URL[p547], Email[p548], or Password[p550] states. (This behavior is included in the used autocorrection state[p899] algorithm below.)

The **`autocorrect`** attribute is an enumerated attribute with the following keywords and states:

Keyword	State	Brief description
`on`	**On**	The user agent is permitted to automatically correct spelling errors while the user types. Whether spelling is automatically corrected while typing left is for the user agent to decide, and may depend on the element as well as the user's preferences.
`off`	**Off**	The user agent is not allowed to automatically correct spelling while the user types.

The attribute's *invalid value default*[p80], *missing value default*[p80], and *empty value default*[p80] are all the On[p899] state.

The **`autocorrect`** getter steps are: return true if the element's used autocorrection state[p899] is On[p899] and false if the element's used autocorrection state[p899] is Off[p899]. The setter steps are: if the given value is true, then the element's `autocorrect`[p899] attribute must be set to "`on`"; otherwise it must be set to "`off`".

To compute the **used autocorrection state** of an element *element*, run these steps:

1. If *element* is an `input`[p539] element whose `type`[p541] attribute is in one of the URL[p547], Email[p548], or Password[p550] states, then return Off[p899].
2. If the `autocorrect`[p899] content attribute is present on *element*, then return the state of the attribute.
3. If *element* is an autocapitalize-and-autocorrect inheriting element[p532] and has a non-null form owner[p626], then return the

state of *element*'s form owner[p626]'s autocorrect[p899] attribute.

4. Return On[p899].

For web developers (non-normative)

***element* . autocorrect[p899]**

Returns the autocorrection behavior of the element. Note that for autocapitalize-and-autocorrect inheriting elements[p532] that inherit their state from a form[p532] element, this will return the autocorrection behavior of the form[p532] element, but for an element in an editable region, this will not return the autocorrection behavior of the editing host[p894] (unless this element is, in fact, the editing host[p894]).

element* . autocorrect[p899] = *value

Updates the autocorrect[p899] content attribute (and thereby changes the autocorrection behavior of the element).

Example

The input[p539] element in the following example would not allow autocorrection, since it does not have an autocorrect[p899] content attribute and therefore inherits from the form[p532] element, which has an attribute of "off[p899]". However, the textarea[p606] element would allow autocorrection, since it has an autocorrect[p899] content attribute with a value of "on[p899]".

```
<form autocorrect="off">
 <input type="search">
 <textarea autocorrect="on"></textarea>
</form>
```

6.8.9 Input modalities: the inputmode[p900] attribute §[p90 0]

User agents can support the inputmode[p900] attribute on form controls (such as the value of textarea[p606] elements), or in elements in an editing host[p894] (e.g., using contenteditable[p892]).

The **inputmode** content attribute is an enumerated attribute[p80] that specifies what kind of input mechanism would be most helpful for users entering content.

Keyword	Description
none	The user agent should not display a virtual keyboard. This keyword is useful for content that renders its own keyboard control.
text	The user agent should display a virtual keyboard capable of text input in the user's locale.
tel	The user agent should display a virtual keyboard capable of telephone number input. This should including keys for the digits 0 to 9, the "#" character, and the "*" character. In some locales, this can also include alphabetic mnemonic labels (e.g., in the US, the key labeled "2" is historically also labeled with the letters A, B, and C).
url	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 "/" and "." characters and for quick input of strings commonly found in domain names such as "www." or ".com".
email	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 "@" character and the "." character.
numeric	The user agent should display a virtual keyboard capable of numeric input. This keyword is useful for PIN entry.
decimal	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.
search	The user agent should display a virtual keyboard optimized for search.

The **inputMode** IDL attribute must reflect[p108] the inputmode[p900] content attribute, limited to only known values[p109].

When inputmode[p900] 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[p541] or pattern[p573] attributes should be used to determine which type of virtual keyboard should be presented to the user.

6.8.10 Input modalities: the enterkeyhint[p901] attribute §[p90 0]

User agents can support the enterkeyhint[p901] attribute on form controls (such as the value of textarea[p606] elements), or in elements in an editing host[p894] (e.g., using contenteditable[p892]).

The **enterkeyhint** content attribute is an enumerated attribute[p80] 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.

Keyword	Description
enter	The user agent should present a cue for the operation 'enter', typically inserting a new line.
done	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.
go	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.
next	The user agent should present a cue for the operation 'next', typically taking the user to the next field that will accept text.
previous	The user agent should present a cue for the operation 'previous', typically taking the user to the previous field that will accept text.
search	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.
send	The user agent should present a cue for the operation 'send', typically delivering the text to its target.

The **enterKeyHint** IDL attribute must reflect[p108] the enterkeyhint[p901] content attribute, limited to only known values[p109].

✔ MDN

When enterkeyhint[p901] 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[p900], type[p541], or pattern[p573] attributes should be used to determine which action label (or icon) to present on the virtual keyboard.

6.9 Find-in-page §[p90] 1

6.9.1 Introduction §[p90] 1

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 the find-in-page[p901] 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[p901] 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[p901], and identifies zero or more **matches**, which are content ranges that satisfy the user query[p901].

One of the matches[p901] is identified to the user as the **active match**. It is highlighted and scrolled into view. The user can navigate through the matches[p901] by advancing the active match[p901] using the find-in-page interface[p901].

Issue #3539 tracks standardizing how find-in-page[p901] underlies the currently-unspecified `window.find()` API.

6.9.2 Interaction with details[p666] and hidden=until-found[p862] §[p90] 1

When find-in-page begins searching for matches, all details[p666] elements in the page which do not have their open[p667] attribute set should have the skipped contents of their second slot become accessible, without modifying the open[p667] attribute, in order to make find-in-page able to search through it. Similarly, all HTML elements with the hidden[p862] attribute in the Hidden Until Found[p862] state should have their skipped contents become accessible without modifying the hidden[p862] attribute in order to make find-in-page able to search through them. After find-in-page finishes searching for matches, the details[p666] elements and the elements with the hidden[p862] attribute in the Hidden Until Found[p862] state should have their contents become skipped again. This entire process must happen synchronously (and so is not observable to users or to author code). [CSSCONTAIN][p1580]

When find-in-page chooses a new active match[p901], perform the following steps:

1. Let *node* be the first node in the active match[p901].
2. Queue a global task[p1195] on the user interaction task source[p1204] given *node*'s relevant global object[p1151] to run the ancestor revealing algorithm[p864] on *node*.

⚠Warning!

When find-in-page auto-expands a `details`[p666] element like this, it will fire a `toggle`[p1576] event. As with the separate `scroll` event that find-in-page fires, this event could be used by the page to discover what the user is typing into the find-in-page dialog. If the page creates a tiny scrollable area with the current search term and every possible next character the user could type separated by a gap, and observes which one the browser scrolls to, it can add that character to the search term and update the scrollable area to incrementally build the search term. By wrapping each possible next match in a closed `details`[p666] element, the page could listen to `toggle`[p1576] events instead of `scroll` events. This attack could be addressed for both events by not acting on every character the user types into the find-in-page dialog.

6.9.3 Interaction with selection §[p90]

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[p901] 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[p901] dismissal or upon a change in the active match[p901] range).

6.10 Close requests and close watchers §[p90]

6.10.1 Close requests §[p90]

In an implementation-defined (and likely device-specific) manner, a user can send a **close request** to the user agent. This indicates that the user wishes to close something that is currently being shown on the screen, such as a popover, menu, dialog, picker, or display mode.

Example

Some example close requests are:

- The Esc key on desktop platforms.
- The back button or gesture on certain mobile platforms such as Android.
- Any assistive technology's dismiss gesture, such as iOS VoiceOver's two-finger scrub "z" gesture.
- A game controller's canonical "back" button, such as the circle button on a DualShock gamepad.

Whenever the user agent receives a potential close request targeted at a `Document`[p135] *document*, it must queue a global task[p1195] on the user interaction task source[p1204] given *document*'s relevant global object[p1151] to perform the following **close request steps**:

1. If *document*'s fullscreen element is not null:
    1. Fully exit fullscreen given *document*'s node navigable[p1036]'s top-level traversable[p1037]'s active document[p1036].
    2. Return.

    Note

    This does not *fire any relevant event, such as `keydown`; it only causes `fullscreenchange` to be eventually fired.*

2. Optionally, skip to the step labeled alternative processing[p903].

    Note

    For example, if the user agent detects user frustration at repeated close request interception by the current web page, it might take this path.

3. Fire any relevant events, per *UI Events* or other relevant specifications. [UIEVENTS][p1587]

Example

An example of a relevant event in the *UI Events* model would be the keydown event that *UI Events* suggests firing when the user presses the Esc key on their keyboard. On most platforms with keyboards, this is treated as a close request[p902], and so would trigger these close request steps[p902].

Example

An example of relevant events that are outside of the model given in *UI Events* would be assistive technology synthesizing an Esc keydown event when the user sends a close request[p902] by using a dismiss gesture.

4. Let *event* be null if no such events are fired, or the Event object representing one of the fired events otherwise. If multiple events are fired, which one is chosen is implementation-defined.
5. If *event* is not null, and its canceled flag is set, then return.
6. If *document* is not fully active[p1051], then return.

Note

This step is necessary because, if event *is not null, then an event listener might have caused* document *to no longer be fully active[p1051].*

7. Let *closedSomething* be the result of processing close watchers[p906] on *document*'s relevant global object[p1151].
8. If *closedSomething* is true, then return.
9. *Alternative processing*: Otherwise, there was nothing watching for a close request[p902]. The user agent may instead interpret this interaction as some other action, instead of interpreting it as a close request.

On platforms where pressing the Esc key is interpreted as a close request[p902], the user agent must interpret the key being pressed *down* as the close request, instead of the key being released. Thus, in the above algorithm, the "relevant events" that are fired must be the single keydown event.

Example

On platforms where Esc is the close request[p902], the user agent will first fire an appropriately-initialized keydown event. If the web developer cancels the event by calling preventDefault(), then nothing further happens. But if the event fires without being canceled, then the user agent proceeds to process close watchers[p906].

Example

On platforms where a back button is a potential close request[p902], no event is involved, so when the back button is pressed, the user agent proceeds directly to process close watchers[p906]. If there is an active[p904] close watcher[p904], then that will get triggered. If there is not, then the user agent can interpret the back button press in another way, for example as a request to traverse the history by a delta[p1077] of −1.

6.10.2 Close watcher infrastructure §[p903]

Each Window[p965] has a **close watcher manager**, which is a struct with the following items:

- **Groups**, a list of lists of close watchers[p904], initially empty.
- **Allowed number of groups**, a number, initially 1.
- **Next user interaction allows a new group**, a boolean, initially true.

Most of the complexity of the close watcher manager[p903] comes from anti-abuse protections designed to prevent developers from disabling users' history traversal abilities, for platforms where a close request[p902]'s fallback action[p903] is the main mechanism of history traversal. In particular:

The grouping of close watchers[p904] is designed so that if multiple close watchers are created without history-action activation[p868], they are grouped together, so that a user-triggered close request[p902] will close all of the close watchers in a group. This ensures that web developers can't intercept an unlimited number of close requests by creating close watchers; instead they can create a number equal to at most 1 + the number of times the user activates the page[p868].

The next user interaction allows a new group[p903] boolean encourages web developers to create close watchers[p904] in a way that is tied to individual user activations[p868]. Without it, each user activation would increase the allowed number of groups[p903], even if the web developer isn't "using" those user activations to create close watchers. In short:

- Allowed: user interaction; create a close watcher in its own group; user interaction; create a close watcher in a second independent group.
- Disallowed: user interaction; user interaction; create a close watcher in its own group; create a close watcher in a second independent group.
- Allowed: user interaction; user interaction; create a close watcher in its own group; create a close watcher grouped with the previous one.

This protection is *not* important for upholding our desired invariant of creating at most (1 + the number of times the user activates the page[p868]) groups. A determined abuser will just create one close watcher per user interaction, "banking" them for future abuse. But this system causes more predictable behavior for the normal case, and encourages non-abusive developers to create close watchers directly in response to user interactions.

To **notify the close watcher manager about user activation** given a `Window`[p965] *window*:

1. Let *manager* be *window*'s close watcher manager[p903].
2. If *manager*'s next user interaction allows a new group[p903] is true, then increment *manager*'s allowed number of groups[p903].
3. Set *manager*'s next user interaction allows a new group[p903] to false.

A **close watcher** is a struct with the following items:

- A **window**, a `Window`[p965].
- A **cancel action**, an algorithm accepting a boolean argument and returning a boolean. The argument indicates whether or not the cancel action algorithm can prevent the close request from proceeding via the algorithm's return value. If the boolean argument is true, then the algorithm can return either true to indicate that the caller will proceed to the close action[p904], or false to indicate that the caller will bail out. If the argument is false, then the return value is always false. This algorithm can never throw an exception.
- A **close action**, an algorithm accepting no arguments and returning nothing. This algorithm can never throw an exception.
- An **is running cancel action** boolean.
- A **get enabled state**, an algorithm accepting no arguments and returning a boolean. This algorithm can never throw an exception.

A close watcher[p904] *closeWatcher* **is active** if *closeWatcher*'s window[p904]'s close watcher manager[p903] contains any list which contains *closeWatcher*.

To **establish a close watcher** given a `Window`[p965] *window*, a list of steps ***cancelAction***, a list of steps ***closeAction***, and an algorithm that returns a boolean ***getEnabledState***:

1. Assert: *window*'s associated `Document`[p966] is fully active[p1051].
2. Let *closeWatcher* be a new close watcher[p904], with

   window[p904]
   window
   cancel action[p904]
   cancelAction
   close action[p904]
   closeAction
   is running cancel action[p904]
   false
   get enabled state[p904]
   getEnabledState

3. Let *manager* be *window*'s close watcher manager[p903].

4. If *manager*'s groups[p903]'s size is less than *manager*'s allowed number of groups[p903], then append « *closeWatcher* » to *manager*'s groups[p903].
5. Otherwise:
    1. Assert: *manager*'s groups[p903]'s size is at least 1 in this branch, since *manager*'s allowed number of groups[p903] is always at least 1.
    2. Append *closeWatcher* to *manager*'s groups[p903]'s last item.
6. Set *manager*'s next user interaction allows a new group[p903] to true.
7. Return *closeWatcher*.

To **request to close** a close watcher[p904] *closeWatcher* with boolean *requireHistoryActionActivation*:

1. If *closeWatcher* is not active[p904], then return true.
2. If the result of running *closeWatcher*'s get enabled state[p904] is false, then return true.
3. If *closeWatcher*'s is running cancel action[p904] is true, then return true.
4. Let *window* be *closeWatcher*'s window[p904].
5. If *window*'s associated `Document`[p966] is not fully active[p1051], then return true.
6. Let *canPreventClose* be true if *requireHistoryActionActivation* is false, or if *window*'s close watcher manager[p903]'s groups[p903]'s size is less than *window*'s close watcher manager[p903]'s allowed number of groups[p903], and *window* has history-action activation[p868]; otherwise false.
7. Set *closeWatcher*'s is running cancel action[p904] to true.
8. Let *shouldContinue* be the result of running *closeWatcher*'s cancel action[p904] given *canPreventClose*.
9. Set *closeWatcher*'s is running cancel action[p904] to false.
10. If *shouldContinue* is false:
    1. Assert: *canPreventClose* is true.
    2. Consume history-action user activation[p869] given *window*.
    3. Return false.

    Note

    Note that since these substeps consume history-action user activation[p869], requesting to close[p905] a close watcher[p904] twice without any intervening user activation[p868] will result in canPreventClose *being false the second time.*
11. Close[p905] *closeWatcher*.
12. Return true.

To **close** a close watcher[p904] *closeWatcher*:

1. If *closeWatcher* is not active[p904], then return.
2. If the result of running *closeWatcher*'s get enabled state[p904] is false, then return.
3. If *closeWatcher*'s window[p904]'s associated `Document`[p966] is not fully active[p1051], then return.
4. Destroy[p905] *closeWatcher*.
5. Run *closeWatcher*'s close action[p904].

To **destroy** a close watcher[p904] *closeWatcher*:

1. Let *manager* be *closeWatcher*'s window[p904]'s close watcher manager[p903].
2. For each *group* of *manager*'s groups[p903]: remove *closeWatcher* from *group*.

3. Remove any item from *manager*'s groups[p903] that is empty.

To **process close watchers** given a Window[p965] *window*:

1. Let *processedACloseWatcher* be false.
2. If *window*'s close watcher manager[p903]'s groups[p903] is not empty:
    1. Let *group* be the last item in *window*'s close watcher manager[p903]'s groups[p903].
    2. For each *closeWatcher* of *group*, in reverse order:
        1. If the result of running *closeWatcher*'s get enabled state[p904] is true, set *processedACloseWatcher* to true.
        2. Let *shouldProceed* be the result of requesting to close[p905] *closeWatcher* with true.
        3. If *shouldProceed* is false, then break.
3. If *window*'s close watcher manager[p903]'s allowed number of groups[p903] is greater than 1, decrement it by 1.
4. Return *processedACloseWatcher*.

6.10.3 The CloseWatcher[p906] interface §[p906]

IDL

```
[Exposed=Window]
interface CloseWatcher : EventTarget {
  constructor(optional CloseWatcherOptions options = {});

  undefined requestClose();
  undefined close();
  undefined destroy();

  attribute EventHandler oncancel;
  attribute EventHandler onclose;
};

dictionary CloseWatcherOptions {
  AbortSignal signal;
};
```

For web developers (non-normative)

***watcher* = new CloseWatcher[p907]()**

***watcher* = new CloseWatcher[p907]({ signal[p906] })**

Creates a new CloseWatcher[p906] instance.

If the signal[p906] option is provided, then *watcher* can be destroyed (as if by watcher.destroy()[p907]) by aborting the given AbortSignal.

If any close watcher[p904] is already active, and the Window[p965] does not have history-action activation[p868], then the resulting CloseWatcher[p906] will be closed together with that already-active close watcher[p904] in response to any close request[p902]. (This already-active close watcher[p904] does not necessarily have to be a CloseWatcher[p906] object; it could be a modal dialog[p675] element, or a popover generated by an element with the popover[p925] attribute.)

***watcher*.requestClose[p907]()**

Acts as if a close request[p902] was sent targeting *watcher*, by first firing a cancel[p1575] event, and if that event is not canceled with preventDefault(), proceeding to fire a close[p1575] event before deactivating the close watcher as if watcher.destroy()[p907] was called.

This is a helper utility that can be used to consolidate cancelation and closing logic into the cancel[p1575] and close[p1575] event handlers, by having all non-close request[p902] closing affordances call this method.

***watcher*.close()**

Immediately fires the close event, and then deactivates the close watcher as if watcher.destroy() was called.

This is a helper utility that can be used trigger the closing logic into the close event handler, skipping any logic in the cancel event handler.

***watcher*.destroy()**

Deactivates *watcher*, so that it will no longer receive close events and so that new independent CloseWatcher instances can be constructed.

This is intended to be called if the relevant UI element is torn down in some other way than being closed.

Each CloseWatcher instance has an **internal close watcher**, which is a close watcher.

The **new CloseWatcher(*options*)** constructor steps are:

1. If this's relevant global object's associated Document is not fully active, then throw an "InvalidStateError" DOMException.
2. Let *closeWatcher* be the result of establishing a close watcher given this's relevant global object, with:
   - *cancelAction* given *canPreventClose* being to return the result of firing an event named cancel at this, with the cancelable attribute initialized to *canPreventClose*.
   - *closeAction* being to fire an event named close at this.
   - *getEnabledState* being to return true.
3. If *options*["signal"] exists:
   1. If *options*["signal"] is aborted, then destroy *closeWatcher*.
   2. Add the following steps to *options*["signal"]:
      1. Destroy *closeWatcher*.
4. Set this's internal close watcher to *closeWatcher*.

The **requestClose()** method steps are to request to close this's internal close watcher with false.

The **close()** method steps are to close this's internal close watcher.

The **destroy()** method steps are to destroy this's internal close watcher.

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 CloseWatcher interface:

Event handler	Event handler event type
oncancel	cancel
onclose	close

¶ 907

Example

If one wanted to implement a custom picker control, which closed itself on a user-provided close request as well as when a close button is pressed, the following code shows how one would use the CloseWatcher API to process close requests:

```
const watcher = new CloseWatcher();
const picker = setUpAndShowPickerDOMElement();

let chosenValue = null;

watcher.onclose = () => {
  chosenValue = picker.querySelector('input').value;
  picker.remove();
```

```
};

picker.querySelector('.close-button').onclick = () => watcher.requestClose();
```

Note how the logic to gather the chosen value is centralized in the CloseWatcher[p906] object's close[p1575] event handler, with the click event handler for the close button delegating to that logic by calling requestClose()[p907].

¶[p908] **Example**

The cancel[p1575] event on CloseWatcher[p906] objects can be used to prevent the close[p1575] event from firing, and the CloseWatcher[p906] from being destroying. A typical use case is as follows:

```
watcher.oncancel = async (e) => {
  if (hasUnsavedData && e.cancelable) {
    e.preventDefault();

    const userReallyWantsToClose = await askForConfirmation("Are you sure you want to close?");
    if (userReallyWantsToClose) {
      hasUnsavedData = false;
      watcher.close();
    }
  }
};
```

For abuse prevention purposes, this event is only cancelable if the page has history-action activation[p868], which will be lost after any given close request[p902]. This ensures that if the user sends a close request twice in a row without any intervening user activation, the request definitely succeeds; the second request ignores any cancel[p1575] event handler's attempt to call preventDefault() and proceeds to close the CloseWatcher[p906].

Combined, the above two examples show how requestClose()[p907] and close()[p907] differ. Because we used requestClose()[p907] in the click event handler for the close button, clicking that button will trigger the CloseWatcher[p906]'s cancel[p1575] event, and thus potentially ask the user for confirmation if there is unsaved data. If we had used close()[p907], then this check would be skipped. Sometimes that is appropriate, but usually requestClose()[p907] is the better option for user-triggered close requests.

¶[p908] **Example**

In addition to the user activation[p868] restrictions for cancel[p1575] events, there is a more subtle form of user activation gating for CloseWatcher[p906] construction. If one creates more than one CloseWatcher[p906] without user activation, then the newly-created one will get grouped together with the most-recently-created close watcher[p904], so that a single close request[p902] will close them both:

```
window.onload = () => {
  // This will work as normal: it is the first close watcher created without user activation.
  (new CloseWatcher()).onclose = () => { /* ... */ };
};

button1.onclick = () => {
  // This will work as normal: the button click counts as user activation.
  (new CloseWatcher()).onclose = () => { /* ... */ };
};

button2.onclick = () => {
  // These will be grouped together, and both will close in response to a single close request.
  (new CloseWatcher()).onclose = () => { /* ... */ };
  (new CloseWatcher()).onclose = () => { /* ... */ };
};
```

This means that calling destroy()[p907], close()[p907], or requestClose()[p907] properly is important. Doing so is the only way to get back the "free" ungrouped close watcher slot. Such close watchers created without user activation are useful for cases like session inactivity timeout dialogs or urgent notifications of server-triggered events, which are not generated in response to user activation.

6.11 Drag and drop §p90 9

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).

6.11.1 Introduction §p90 9

This section is non-normative.

To make an element draggable, give the element a `draggable`p924 attribute, and set an event listener for `dragstart`p923 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`p911 object and set the allowed effects (copy, move, link, or some combination).

For example:

```
<p>What fruits do you like?</p>
<ol ondragstart="dragStartHandler(event)">
 <li draggable="true" 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`p924 event handler reports whether or not the drop target is potentially willing to accept the drop, by canceling the event.

2. The `dragover`p924 event handler specifies what feedback will be shown to the user, by setting the `dropEffect`p913 attribute of the `DataTransfer`p911 associated with the event. This event also needs to be canceled.

3. The `drop`p924 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`p913 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';
    } else {
      li.textContent = 'Unknown Fruit';
    }
    event.target.appendChild(li);
  }
</script>
```

To remove the original element (the one that was dragged) from the display, the dragend[p924] event can be used.

For our example here, that means updating the original markup to handle that event:

```
<p>What fruits do you like?</p>
<ol ondragstart="dragStartHandler(event)" ondragend="dragEndHandler(event)">
 ...as before...
</ol>
<script>
  function dragStartHandler(event) {
    // ...as before...
  }
  function dragEndHandler(event) {
    if (event.dataTransfer.dropEffect == 'move') {
      // remove the dragged element
      event.target.parentNode.removeChild(event.target);
    }
  }
</script>
```

6.11.2 The drag data store §[p91]

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 filename.

  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[p911].

  The actual data

  A Unicode or binary string, in some cases with a filename (itself a Unicode string), as per the drag data item kind[p911].

  The drag data store item list[p911] is ordered in the order that the items were added to the list; most recently added last.

- The following information, used to generate the UI feedback during the drag:

  - User-agent-defined default feedback information, known as the **drag data store default feedback**.
  - Optionally, a bitmap image and the coordinate of a point within that image, known as the **drag data store bitmap** and **drag data store hot spot coordinate**.

- A **drag data store mode**, which is one of the following:

  Read/write mode

  For the `dragstart`[p923] event. New data can be added to the drag data store[p911].

  Read-only mode

  For the `drop`[p924] 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[p911] 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[p911] is **created**, it must be initialized such that its drag data store item list[p911] is empty, it has no drag data store default feedback[p911], it has no drag data store bitmap[p911] and drag data store hot spot coordinate[p911], its drag data store mode[p911] is protected mode[p911], and its drag data store allowed effects state[p911] is the string "`uninitialized`[p913]".

6.11.3 The `DataTransfer`[p911] interface §[p91]

`DataTransfer`[p911] objects are used to expose the drag data store[p911] 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[p913]()**

Creates a new DataTransfer[p911] object with an empty drag data store[p911].

***dataTransfer*.dropEffect[p913] [= *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[p913] attribute, then the operation will fail.

Can be set, to change the selected operation.

The possible values are "none[p913]", "copy[p913]", "link[p913]", and "move[p913]".

***dataTransfer*.effectAllowed[p913] [= *value*]**

Returns the kinds of operations that are to be allowed.

Can be set (during the dragstart[p923] event), to change the allowed operations.

The possible values are "none[p913]", "copy[p913]", "copyLink[p913]", "copyMove[p913]", "link[p913]", "linkMove[p913]", "move[p913]", "all[p913]", and "uninitialized[p913]",

***dataTransfer*.items[p913]**

Returns a DataTransferItemList[p915] object, with the drag data.

***dataTransfer*.setDragImage[p913](*element*, *x*, *y*)**

Uses the given element to update the drag feedback, replacing any previously specified feedback.

***dataTransfer*.types[p913]**

Returns a frozen array listing the formats that were set in the dragstart[p923] event. In addition, if any files are being dragged, then one of the types will be the string "Files".

***data* = *dataTransfer*.getData[p913](*format*)**

Returns the specified data. If there is no such data, returns the empty string.

***dataTransfer*.setData[p914](*format*, *data*)**

Adds the specified data.

***dataTransfer*.clearData[p914]([*format*])**

Removes the data of the specified formats. Removes all data if the argument is omitted.

***dataTransfer*.files[p914]**

Returns a FileList of the files being dragged, if any.

DataTransfer[p911] objects that are created as part of drag-and-drop events[p923] are only valid while those events are being fired.

A DataTransfer[p911] object is associated with a drag data store[p911] while it is valid.

A DataTransfer[p911] object has an associated **types array**, which is a FrozenArray<DOMString>, initially empty. When the contents of the DataTransfer[p911] object's drag data store item list[p911] change, or when the DataTransfer[p911] object becomes no longer associated with a drag data store[p911], run the following steps:

1. Let *L* be an empty sequence.

2. If the DataTransfer[p911] object is still associated with a drag data store[p911]:

    1. For each item in the DataTransfer[p911] object's drag data store item list[p911] whose kind[p911] is *text*, add an entry to *L* consisting of the item's type string[p911].

    2. If there are any items in the DataTransfer[p911] object's drag data store item list[p911] whose kind[p911] 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[p911] object's types array[p912] to the result of creating a frozen array from *L*.

The **`DataTransfer()`** constructor, when invoked, must return a newly created DataTransfer[p911] object initialized as follows:

1. Set the drag data store[p911]'s item list[p911] to be an empty list.

2. Set the drag data store[p911]'s mode[p911] to read/write mode[p911].

3. Set the dropEffect[p913] and effectAllowed[p913] 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[p911] object is created, the dropEffect[p913] 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[p913] attribute during the dragenter[p924] and dragover[p924] events. When the DataTransfer[p911] object is created, the effectAllowed[p913] attribute is set to a string value. On getting, it must return its current value. On setting, if the drag data store[p911]'s mode[p911] is the read/write mode[p911] 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[p915] object associated with the DataTransfer[p911] object.

The **`setDragImage(image, x, y)`** method must run the following steps:

1. If the DataTransfer[p911] object is no longer associated with a drag data store[p911], return. Nothing happens.

2. If the drag data store[p911]'s mode[p911] is not the read/write mode[p911], return. Nothing happens.

3. If *image* is an img[p359] element, then set the drag data store bitmap[p911] to the element's image (at its natural size); otherwise, set the drag data store bitmap[p911] 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[p911] to the given *x*, *y* coordinate.

The **`types`** attribute must return this DataTransfer[p911] object's types array[p912].

The **`getData(format)`** method must run the following steps:

1. If the DataTransfer[p911] object is no longer associated with a drag data store[p911], then return the empty string.

2. If the drag data store[p911]'s mode[p911] is the protected mode[p911], 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[p911] whose kind[p911] is *text* and whose type string[p911] is equal to *format*, return the empty string.

8. Let *result* be the data of the item in the drag data store item list[p911] whose kind[p911] is *Plain Unicode string* and whose type string[p911] is equal to *format*.

9. If *convert-to-URL* is true, then parse *result* as appropriate for `text/uri-list` data, and then set *result* to the first URL from the list, if any, or the empty string otherwise. [RFC2483][p1585]

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(format)`** 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. Set *format* to *format*, 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.

6.11.3.1 The `DataTransferItemList` interface

Each `DataTransfer` object is associated with a `DataTransferItemList` object.

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**[p915]

Returns the number of items in the drag data store[p911].

***items*[*index*]**

Returns the DataTransferItem[p916] object representing the *index*th entry in the drag data store[p911].

***items*.remove**[p916]**(*index*)**

Removes the *index*th entry in the drag data store[p911].

***items*.clear**[p916]**()**

Removes all the entries in the drag data store[p911].

***items*.add**[p915]**(*data*)**

***items*.add**[p915]**(*data*, *type*)**

Adds a new entry for the given data to the drag data store[p911]. If the data is plain text then a *type* string has to be provided also.

While the DataTransferItemList[p915] object's DataTransfer[p911] object is associated with a drag data store[p911], the DataTransferItemList[p915] object's *mode* is the same as the drag data store mode[p911]. When the DataTransferItemList[p915] object's DataTransfer[p911] object is *not* associated with a drag data store[p911], the DataTransferItemList[p915] object's *mode* is the *disabled mode*. The drag data store[p911] referenced in this section (which is used only when the DataTransferItemList[p915] object is not in the *disabled mode*) is the drag data store[p911] with which the DataTransferItemList[p915] object's DataTransfer[p911] 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[p911].

When a DataTransferItemList[p915] object is not in the *disabled mode*, its supported property indices are the indices of the drag data store item list[p911].

To determine the value of an indexed property *i* of a DataTransferItemList[p915] object, the user agent must return a DataTransferItem[p916] object representing the *i*th item in the drag data store[p911]. The same object must be returned each time a particular item is obtained from this DataTransferItemList[p915] object. The DataTransferItem[p916] object must be associated with the same DataTransfer[p911] object as the DataTransferItemList[p915] object when it is first created.

The **add()** method must run the following steps:

1. If the DataTransferItemList[p915] object is not in the *read/write mode*[p911], 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[p911] whose kind[p911] is *text* and whose type string[p911] is equal to the value of the method's second argument, converted to ASCII lowercase, then throw a "NotSupportedError" DOMException.

   Otherwise, add an item to the drag data store item list[p911] whose kind[p911] is *text*, whose type string[p911] 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[p911] whose kind[p911] is *File*, whose type string[p911] 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[p915] corresponding to the newly added item, and return that value (a newly created DataTransferItem[p916] object).

The **remove(*index*)** method must run these steps:

1. If the DataTransferItemList[p915] object is not in the *read/write mode*[p911], throw an "InvalidStateError" DOMException.
2. If the drag data store[p911] does not contain an *index*th item, then return.
3. Remove the *index*th item from the drag data store[p911].

The **clear()** method, if the DataTransferItemList[p915] object is in the *read/write mode*[p911], must remove all the items from the drag data store[p911]. Otherwise, it must do nothing.

6.11.3.2 The DataTransferItem[p916] interface §[p916]

Each DataTransferItem[p916] object is associated with a DataTransfer[p911] 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**[p916]

Returns the drag data item kind[p911], one of: "string", "file".

***item*.type**[p916]

Returns the drag data item type string[p911].

***item*.getAsString**[p916]**(*callback*)**

Invokes the callback with the string data as the argument, if the drag data item kind[p911] is *text*.

***file* = *item*.getAsFile**[p917]**()**

Returns a File object, if the drag data item kind[p911] is *File*.

While the DataTransferItem[p916] object's DataTransfer[p911] object is associated with a drag data store[p911] and that drag data store[p911]'s drag data store item list[p911] still contains the item that the DataTransferItem[p916] object represents, the DataTransferItem[p916] object's *mode* is the same as the drag data store mode[p911]. When the DataTransferItem[p916] object's DataTransfer[p911] object is *not* associated with a drag data store[p911], or if the item that the DataTransferItem[p916] object represents has been removed from the relevant drag data store item list[p911], the DataTransferItem[p916] object's *mode* is the *disabled mode*. The drag data store[p911] referenced in this section (which is used only when the DataTransferItem[p916] object is not in the *disabled mode*) is the drag data store[p911] with which the DataTransferItem[p916] object's DataTransfer[p911] object is associated.

The **kind** attribute must return the empty string if the DataTransferItem[p916] 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[p911] of the item represented by the DataTransferItem[p916] object:

Kind	String
Text	"string"
File	"file"

The **type** attribute must return the empty string if the DataTransferItem[p916] object is in the *disabled mode*; otherwise it must return the drag data item type string[p911] of the item represented by the DataTransferItem[p916] object.

The **getAsString(*callback*)** method must run the following steps:

1. If the *callback* is null, return.
2. If the DataTransferItem[p916] object is not in the *read/write mode*[p911] or the *read-only mode*[p911], return. The callback is never invoked.
3. If the drag data item kind[p911] is not *text*, then return. The callback is never invoked.
4. Otherwise, queue a task[p1194] to invoke *callback*, passing the actual data of the item represented by the DataTransferItem[p916] object as the argument.

The **getAsFile()** method must run the following steps:

1. If the DataTransferItem[p916] object is not in the *read/write mode*[p911] or the *read-only mode*[p911], then return null.
2. If the drag data item kind[p911] is not *File*, then return null.
3. Return a new File object representing the actual data of the item represented by the DataTransferItem[p916] object.

6.11.4 The DragEvent[p917] interface §[p917]

The drag-and-drop processing model involves several events. They all use the DragEvent[p917] interface.

IDL

```
[Exposed=Window]
interface DragEvent : MouseEvent {
  constructor(DOMString type, optional DragEventInit eventInitDict = {});

  readonly attribute DataTransfer? dataTransfer;
};

dictionary DragEventInit : MouseEventInit {
  DataTransfer? dataTransfer = null;
};
```

For web developers (non-normative)

***event*.dataTransfer**[p917]

Returns the DataTransfer[p911] object for the event.

Note

Although, for consistency with other event interfaces, the DragEvent[p917] interface has a constructor, it is not particularly useful. In particular, there's no way to create a useful DataTransfer[p911] object from script, as DataTransfer[p911] objects have a processing and security model that is coordinated by the browser during drag-and-drops.

The **dataTransfer** attribute of the DragEvent[p917] 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[p911], 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[p1151] of the Document[p135] object of the specified target element.
4. If *e* is dragstart[p923], then set the drag data store mode[p911] to the read/write mode[p911] and set *dataDragStoreWasChanged* to true.

   If *e* is drop[p924], set the drag data store mode[p911] to the read-only mode[p911].
5. Let *dataTransfer* be a newly created DataTransfer[p911] object associated with the given drag data store[p911].
6. Set the effectAllowed[p913] attribute to the drag data store[p911]'s drag data store allowed effects state[p911].

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):

effectAllowed	dropEffect
"none"	"none"
"copy"	"copy"
"copyLink"	"copy", or, if appropriate, "link"
"copyMove"	"copy", or, if appropriate, "move"
"all"	"copy", or, if appropriate, either "link" or "move"
"link"	"link"
"linkMove"	"link", or, if appropriate, "move"
"move"	"move"
"uninitialized" when what is being dragged is a selection from a text control	"move", or, if appropriate, either "copy" or "link"
"uninitialized" when what is being dragged is a selection	"copy", or, if appropriate, either "link" or "move"
"uninitialized" when what is being dragged is an a element with an href attribute	"link", or, if appropriate, either "copy" or "move"
Any other case	"copy", or, if appropriate, either "link" or "move"

Where the table above provides **possibly appropriate alternatives**, user agents may instead use the listed alternative values if platform conventions dictate that the user has requested those alternate effects.

Example

For example, Windows platform conventions are such that dragging while holding the "alt" key indicates a preference for linking the data, rather than moving or copying it. Therefore, on a Windows system, if "link" is an option according to the table above while the "alt" key is depressed, the user agent could select that instead of "copy" or "move".

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 dragend, then initialize *event*'s cancelable attribute to true.

11. Initialize *event*'s mouse and key attributes 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 mode back to the protected mode.

15. Break the association between *dataTransfer* and the drag data store.

6.11.5 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[p135], it is the first element, going up the ancestor chain, starting at the node that the user tried to drag, that has the IDL attribute draggable[p924] 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.

Note

img[p359] elements and a[p267] elements with an href[p314] attribute have their draggable[p924] attribute set to true by default.

2. Create a drag data store[p911]. All the DND events fired subsequently by the steps in this section must use this drag data store[p911].

3. Establish which DOM node is the **source node**, as follows:

   If it is a selection that is being dragged, then the source node[p919] 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[p919] 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[p919] is the element that is being dragged.

   Otherwise, the source node[p919] is part of another document or application. When this specification requires that an event be dispatched at the source node[p919] in this case, the user agent must instead follow the platform-specific conventions relevant to that situation.

   Note

   Multiple events are fired on the source node[p919] 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[p919] 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[p919] contains only the source node[p919], if any.

5. If it is a selection that is being dragged, then add an item to the drag data store item list[p911], with its properties set as follows:

   The drag data item type string[p911]

   "text/plain"

   The drag data item kind[p911]

   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[p911], with their properties set as follows:

   The drag data item type string[p911]

   The MIME type of the file, if known, or "application/octet-stream" otherwise.

   The drag data item kind[p911]

   File

   The actual data

   The file's contents and name.

   Note

   Dragging files can currently only happen from outside a navigable[p1035], 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[p911] 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[p911] 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[p919] is not empty, then extract the microdata from those nodes into a JSON form[p859], and add one item to the drag data store item list[p911], with its properties set as follows:

   The drag data item type string[p911]
   `application/microdata+json`[p1550]

   The drag data item kind[p911]
   Text

   The actual data
   The resulting JSON string.

7. Run the following substeps:

   1. Let *urls* be « ».
   2. For each *node* in the list of dragged nodes[p919]:

      If the node is an `a`[p267] element with an `href`[p314] attribute
      Add to *urls* the result of encoding-parsing-and-serializing a URL[p101] given the element's `href`[p314] content attribute's value, relative to the element's node document.

      If the node is an `img`[p359] element with a `src`[p360] attribute
      Add to *urls* the result of encoding-parsing-and-serializing a URL[p101] given the element's `src`[p360] content attribute's value, relative to the element's node 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[p911], with its properties set as follows:

      The drag data item type string[p911]
      `text/uri-list`[p1578]

      The drag data item kind[p911]
      Text

      The actual data
      url string

8. Update the drag data store default feedback[p911] 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[p917] named `dragstart`[p923] at the source node[p919].

   If the event is canceled, then the drag-and-drop operation should not occur; return.

   Note
   Since events with no event listeners registered are, almost by definition, never canceled, drag-and-drop is always available to the user if the author does not specifically prevent it.

10. Fire a pointer event at the source node[p919] named `pointercancel`, and fire any other follow-up events as required by *Pointer Events*. [POINTEREVENTS][p1585]

11. Initiate the drag-and-drop operation[p921] 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[p911], if any. In this case, the drag data store hot spot coordinate[p911] 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.

        If the event is canceled, then set the current target element to the immediate user selection.

        Otherwise, run the appropriate step from the following list:

        ↪ **If the immediate user selection is a text 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 "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 state) or an editing host or editable element, and the drag data store item list has an item with the drag data item type string "`text/plain`" and the drag data item kind *text***

   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:

`effectAllowed`	`dropEffect`	Drag operation
"`uninitialized`", "`copy`", "`copyLink`", "`copyMove`", or "`all`"	"`copy`"	"`copy`"
"`uninitialized`", "`link`", "`copyLink`", "`linkMove`", or "`all`"	"`link`"	"`link`"
"`uninitialized`", "`move`", "`copyMove`", "`linkMove`", or "`all`"	"`move`"	"`move`"
Any other case		"`none`"

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:

Drag operation	Feedback
"`copy`"	Data will be copied if dropped here.
"`link`"	Data will be linked if dropped here.
"`move`"	Data will be moved if dropped here.
"`none`"	No operation allowed, dropping here will cancel the drag-and-drop operation.

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 "`text/plain`" 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.

6.11.6 Events summary §

This section is non-normative.

The following events are involved in the drag-and-drop model.

Event name	Target	Cancelable?	Drag data store mode	`dropEffect`	Default Action
`dragstart`	Source node	✓ Cancelable	Read/write mode	"none"	Initiate the drag-and-drop operation
`drag`	Source node	✓ Cancelable	Protected mode	"none"	Continue the drag-and-drop operation

Event name	Target	Cancelable?	Drag data store mode	dropEffect	Default Action
dragenter	Immediate user selection or the body element	✓ Cancelable	Protected mode	Based on effectAllowed value	Reject immediate user selection as potential target element
dragleave	Previous target element	—	Protected mode	"none"	None
dragover	Current target element	✓ Cancelable	Protected mode	Based on effectAllowed value	Reset the current drag operation to "none"
drop	Current target element	✓ Cancelable	Read-only mode	Current drag operation	Varies
dragend	Source node	—	Protected mode	Current drag operation	Varies

All of 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.

6.11.7 The draggable attribute

All HTML elements may have the **draggable** content attribute set. The draggable attribute is an enumerated attribute with the following keywords and states:

Keyword	State	Brief description
true	**True**	The element will be draggable.
false	**False**	The element will not be draggable.

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

For web developers (non-normative)

***element*.draggable [= *value*]**

Returns true if the element is draggable; otherwise, returns false.

Can be set, to override the default and set the draggable content attribute.

The **draggable** IDL attribute, whose value depends on the content attribute's in the way described below, controls whether or not the element is draggable. Generally, only text selections are draggable, but elements whose draggable IDL attribute is true become draggable as well.

If an element's draggable content attribute has the state True, the draggable IDL attribute must return true.

Otherwise, if the element's draggable content attribute has the state False, the draggable IDL attribute must return false.

Otherwise, the element's draggable content attribute has the state Auto. If the element is an img element, an object element that represents an image, or an a element with an href content attribute, the draggable IDL attribute must return true; otherwise, the draggable IDL attribute must return false.

If the draggable IDL attribute is set to the value false, the draggable content attribute must be set to the literal value "false". If the draggable IDL attribute is set to the value true, the draggable content attribute must be set to the literal value "true".

6.11.8 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 drag operation — if any scripts end the drag operation, it must be considered unsuccessful (canceled) and the drop event must not be fired.

User agents should take care to not start drag-and-drop operations in response to script actions. For example, in a mouse-and-window environment, if a script moves a window while the user has 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.

6.12 The popover attribute

All HTML elements may have the **popover** content attribute set. When specified, the element won't be rendered until it becomes shown, at which point it will be rendered on top of other page content.

Example

The popover attribute is a global attribute that allows authors flexibility to ensure popover functionality can be applied to elements with the most relevant semantics.

The following demonstrates how one might create a popover sub-navigation list of links, within the global navigation for a website.

```
<ul>
  <li>
    <a href=...>All Products</a>
    <button popovertarget=sub-nav>
     <img src=down-arrow.png alt="Product pages">
    </button>
    <ul popover id=sub-nav>
     <li><a href=...>Shirts</a>
     <li><a href=...>Shoes</a>
     <li><a href=...>Hats etc.</a>
    </ul>
  </li>
  <!-- other list items and links here -->
</ul>
```

When using popover on elements without accessibility semantics, for instance the div element, authors should use the appropriate ARIA attributes to ensure the popover is accessible.

Example

The following shows the baseline markup to create a custom menu popover, where the first menuitem will receive keyboard focus when the popover is invoked due to the use of the `autofocus` attribute. Navigating the menuitems with arrow keys and activation behaviors would still need author scripting. Additional requirements for building custom menus widgets are defined in the WAI-ARIA specification.

```
<button popovertarget=m>Actions</button>
<div role=menu id=m popover>
  <button role=menuitem tabindex=-1 autofocus>Edit</button>
  <button role=menuitem tabindex=-1>Hide</button>
```

```
  <button role=menuitem tabindex=-1>Delete</button>
</div>
```

A popover can be useful for rendering a status message, confirming the action performed by the user. The following demonstrates how one could reveal a popover in an output[p611] element.

```
<button id=submit>Submit</button>
<p><output><span popover=manual></span></output></p>

<script>
 const sBtn = document.getElementById("submit");
 const outSpan = document.querySelector("output [popover=manual]");
 let successMessage;
 let errorMessage;

 /* define logic for determining success of action
  and determining the appropriate success or error
  messages to use */

 sBtn.addEventListener("click", ()=> {
  if ( success ) {
   outSpan.textContent = successMessage;
  }
  else {
   outSpan.textContent = errorMessage;
  }
  outSpan.showPopover();

  setTimeout(function () {
   outSpan.hidePopover();
  }, 10000);
 });
</script>
```

Note

Inserting a popover element into an output[p611] element will generally cause screen readers to announce the content when it becomes visible. Depending on the complexity or frequency of the content, this could be either useful or annoying to users of these assistive technologies. Keep this in mind when using the output[p611] element or other ARIA live regions to ensure the best user experience.

The popover[p925] attribute is an enumerated attribute[p80] with the following keywords and states:

Keyword	State	Brief description
auto	**Auto**	Closes other popovers when opened; has light dismiss[p937] and responds to close requests[p902].
manual	**Manual**	Does not close other popovers; does not light dismiss[p937] or respond to close requests[p902].
hint	**Hint**	Closes other hint popovers when opened, but not other auto popovers; has light dismiss[p937] and responds to close requests[p902].

The attribute's *missing value default*[p80] is the **No Popover** state, its *invalid value default*[p80] is the Manual[p926] state, and its *empty value default*[p80] is the Auto[p926] state.

The **popover** IDL attribute must reflect[p108] the popover[p925] attribute, limited to only known values[p109].

Every HTML element[p46] has a **popover visibility state**, initially hidden[p926], with these potential values:

- **hidden**
- **showing**

Every Document[p135] has a **popover pointerdown target**, which is an HTML element[p46] or null, initially null.

Every HTML element[p46] has a **popover trigger**, which is an HTML element[p46] or null, initially set to null.

Every HTML element[p46] has a **popover hiding**, which is a boolean, initially set to false.

Every HTML element[p46] **popover toggle task tracker**, which is a toggle task tracker[p872] or null, initially null.

Every HTML element[p46] has a **popover close watcher**, which is a close watcher[p904] or null, initially null.

Every HTML element[p46] has an **opened in popover mode**, which is "auto", "hint", or null, initially null.

Every Document[p135] has a **hiding popover nesting count**, which is a number, initially 0.

Every Document[p135] has a **showing popover**, which is a boolean, initially false.

Every Document[p135] has a **hint stack parent**, which is an HTML element[p46] or null, initially null.

Note

The hint stack parent[p927] tracks which item in the showing auto popover list[p934] is the 'parent' of the first item in the showing hint popover list[p935], or null if the first item in showing hint popover list[p935] is not 'child' to an item in the showing auto popover list[p934].

Therefore, when the hint stack parent[p927] is not null, it will have an opened in popover mode[p927] of "auto".

The following attribute change steps, given *element*, *localName*, *oldValue*, *value*, and *namespace*, are used for all HTML elements[p46]:

1. If *namespace* is not null, then return.
2. If *localName* is not popover[p925], then return.
3. If *element*'s popover visibility state[p926] is in the showing state[p926] and *oldValue* and *value* are in different states[p925], then run the hide popover algorithm[p930] given *element*, true, true, false, and null.

For web developers (non-normative)

***element*.showPopover[p927]()**

Shows the popover *element* by adding it to the top layer. If *element*'s popover[p925] attribute is in the Auto[p926] state, then this will also close all other Auto[p926] popovers unless they are an ancestor of *element* according to the topmost popover ancestor[p933] algorithm.

***element*.hidePopover[p930]()**

Hides the popover *element* by removing it from the top layer and applying `display: none` to it.

***element*.togglePopover[p931]()**

If the popover *element* is not showing, then this method shows it. Otherwise, this method hides it. This method returns true if the popover is open after calling it, otherwise false.

The **showPopover(*options*)** method steps are:

1. Let *source* be *options*["source[p150]"] if it exists; otherwise, null.
2. Run show popover[p927] given this, true, and *source*.

To **show popover**, given an HTML element[p46] *element*, a boolean *throwExceptions*, and an HTML element[p46] or null *source*:

1. Let *document* be *element*'s node document.
2. If *document*'s showing popover[p927] is true, or *document*'s hiding popover nesting count[p927] is not 0:
    1. If *throwExceptions* is true, then throw a "InvalidStateError" DOMException.
    2. Return.

    Note

    This prevents showing a popover during the show or hide of another popover.
3. Let *validityResult* be the result of running check popover validity[p934] given *element*, false, and null. If this throws an exception, catch it, and set *validityResult* to that exception.

4. If *validityResult* is not true:
    1. If *throwExceptions* is true and *validityResult* is a DOMException, then throw *validityResult*.
    2. Return.
5. Assert: *element*'s popover trigger is null.
6. Assert: *element* is not in *document*'s top layer.
7. Set *document*'s showing popover to true.
8. Let *cleanupShowingSteps* be the following steps:
    1. Set *document*'s showing popover to false.
9. If the result of firing an event named beforetoggle, using ToggleEvent, with the cancelable attribute initialized to true, the oldState attribute initialized to "closed", the newState attribute initialized to "open", and the source attribute initialized to *source* at *element* is false, then run *cleanupShowingSteps* and return.
10. Set *validityResult* to the result of running check popover validity given *element*, false, and *document*. If this throws an exception, catch it, and set *validityResult* to that exception.

    Note

    Check popover validity is called again because firing the beforetoggle event could have disconnected this element or changed its popover attribute.
11. If *validityResult* is not true:
    1. Run *cleanupShowingSteps*.
    2. If *throwExceptions* is true and *validityResult* is a DOMException, then throw *validityResult*.
    3. Return.
12. Let *shouldRestoreFocus* be false.
13. Let *originalType* be the current state of *element*'s popover attribute.
14. Let *effectiveType* be *originalType*.
15. If *originalType* is Auto or Hint:
    1. Let *ancestor* be the result of running the topmost popover ancestor algorithm given *element*, *source*, and true.
    2. If all of the following are true:
        - *ancestor* is not null;
        - *ancestor*'s opened in popover mode is "hint"; and
        - *effectiveType* is the Auto state,

        then set *effectiveType* to the Hint state.

        Note

        Hint popovers are lower priority than Auto popovers, so an Auto popover cannot have a Hint popover as a 'parent'. To resolve this case, the effectiveType *is 'downgraded' to Hint.*
    3. Run hide popover stack until given *document*, *ancestor*, Hint, *shouldRestoreFocus*, and true.
    4. If *effectiveType* is the Auto state, then run hide popover stack until given *document*, *ancestor*, Auto, *shouldRestoreFocus*, and true.
    5. If *originalType* is not equal to the value of *element*'s popover attribute:
        1. Run *cleanupShowingSteps*.
        2. If *throwExceptions* is true, then throw an "InvalidStateError" DOMException.
        3. Return.

6. Set *validityResult* to the result of running check popover validity[p934] given *element*, false, and *document*. If this throws an exception, catch it, and set *validityResult* to that exception.

   Note

   Check popover validity[p934] is called again because running hide popover stack until[p932] above could have fired the `beforetoggle`[p1574] event, and an event handler could have disconnected this element or changed its `popover`[p925] attribute.

7. If *validityResult* is not true:

   1. Run *cleanupShowingSteps*.
   2. If *throwExceptions* is true and *validityResult* is a `DOMException`, then throw *validityResult*.
   3. Return.

8. If the result of running topmost auto or hint popover[p934] on *document* is null, then set *shouldRestoreFocus* to true.

   Note

   This ensures that focus is returned to the previously-focused element only for the first popover in a stack.

9. If *effectiveType* is Auto[p926]:

   1. Assert: *document*'s showing auto popover list[p934] does not contain *element*.
   2. Set *element*'s opened in popover mode[p927] to "`auto`".

   Otherwise:

   1. Assert: *effectiveType* is Hint[p926].
   2. Assert: *document*'s showing hint popover list[p935] does not contain *element*.
   3. Set *element*'s opened in popover mode[p927] to "`hint`".

10. Set *element*'s popover close watcher[p927] to the result of establishing a close watcher[p904] given *element*'s relevant global object[p1151], with:

    - *cancelAction*[p904] being to return true.
    - *closeAction*[p904] being to hide a popover[p930] given *element*, true, true, false, and null.
    - *getEnabledState*[p904] being to return true.

16. Set *element*'s previously focused element[p679] to null.
17. Let *originallyFocusedElement* be *document*'s focused area of the document[p875]'s DOM anchor[p874].
18. Add an element to the top layer given *element*.
19. If *effectiveType* is Hint[p926] and *ancestor*'s opened in popover mode[p927] is "`auto`", then set *document*'s hint stack parent[p927] to *ancestor*.
20. Set *element*'s popover visibility state[p926] to showing[p926].
21. Set *element*'s popover trigger[p927] to *source*.
22. Set *element*'s implicit anchor element to *source*.
23. Run the popover focusing steps[p934] given *element*.
24. If *shouldRestoreFocus* is true and *element*'s `popover`[p925] attribute is not in the No Popover[p926] state, then set *element*'s previously focused element[p679] to *originallyFocusedElement*.
25. Run *cleanupShowingSteps*.
26. Queue a popover toggle event task[p929] given *element*, "`closed`", "`open`", and *source*.

To **queue a popover toggle event task** given an element *element*, a string *oldState*, a string *newState*, and an `Element` or null *source*:

1. If *element*'s popover toggle task tracker[p927] is not null:
    1. Set *oldState* to *element*'s popover toggle task tracker[p927]'s old state[p872].
    2. Remove *element*'s popover toggle task tracker[p927]'s task[p872] from its task queue[p1193].
    3. Set *element*'s popover toggle task tracker[p927] to null.
2. Queue an element task[p1195] given the DOM manipulation task source[p1203] and *element* to run the following steps:
    1. Fire an event named `toggle`[p1576] at *element*, using `ToggleEvent`[p871], with the `oldState`[p872] attribute initialized to *oldState*, the `newState`[p872] attribute initialized to *newState*, and the `source`[p872] attribute initialized to *source*.
    2. Set *element*'s popover toggle task tracker[p927] to null.
3. Set *element*'s popover toggle task tracker[p927] to a struct with task[p872] set to the just-queued task[p1194] and old state[p872] set to *oldState*.

The **`hidePopover()`** method steps are: ✔ MDN

1. Run the hide popover algorithm[p930] given this, true, true, true, and null.

To **hide a popover** given an HTML element[p46] *element*, a boolean *focusPreviousElement*, a boolean *fireEvents*, a boolean *throwExceptions*, and an HTML element[p46] or null *source*:

1. Let *validityResult* be the result of running check popover validity[p934] given *element*, true, and null. If this throws an exception, catch it, and let *validityResult* be that exception.
2. If *validityResult* is not true:
    1. If *throwExceptions* is true and *validityResult* is a `DOMException`, then throw *validityResult*.
    2. Return.
3. Let *document* be *element*'s node document.
4. Let *nestedHide* be *element*'s popover hiding[p927].
5. Set *element*'s popover hiding[p927] to true.
6. If *nestedHide* is true, then set *fireEvents* to false.
7. Increment *document*'s hiding popover nesting count[p927].
8. Let *cleanupSteps* be the following steps:
    1. If *nestedHide* is false, then set *element*'s popover hiding[p927] to false.
    2. If *element*'s popover close watcher[p927] is not null:
        1. Destroy[p905] *element*'s popover close watcher[p927].
        2. Set *element*'s popover close watcher[p927] to null.
    3. Decrement *document*'s hiding popover nesting count[p927].
9. Let *autoPopoverListContainsElement* be true if *document*'s showing auto popover list[p934] contains *element*; otherwise false.
10. Let *hintPopoverListContainsElement* be true if *document*'s showing hint popover list[p935] contains *element*; otherwise false.
11. If *element*'s opened in popover mode[p927] is "`auto`" or "`hint`":
    1. If *hintPopoverListContainsElement* is true, then run hide popover stack until[p932] given *document*, *element*, Hint[p926], *focusPreviousElement*, and *fireEvents*.
    2. If *element* is *document*'s hint stack parent[p927], then run hide popover stack until[p932] given *document*, null, Hint[p926], *focusPreviousElement*, and *fireEvents*.

> **Note**
> *If the* document's *hint stack parent*[p927] *is being hidden, then* all *hint popovers are hidden.*

3. If *autoPopoverListContainsElement* is true, then run hide popover stack until[p932] given *document*, *element*, Auto[p926], *focusPreviousElement*, and *fireEvents*.

4. Set *validityResult* to the result of running check popover validity[p934] given *element*, true, and null. If this throws an exception, catch it, and set *validityResult* to that exception.

   Note

   Check popover validity[p934] is called again because running hide popover stack until[p932] could have disconnected element *or changed its* popover[p925] *attribute.*

5. If *validityResult* is not true:

   1. Run *cleanupSteps*.
   2. If *throwExceptions* is true and *validityResult* is a DOMException, then throw *validityResult*.
   3. Return.

12. If *fireEvents* is true:

    1. Fire an event named beforetoggle[p1574], using ToggleEvent[p871], with the oldState[p872] attribute initialized to "open", the newState[p872] attribute initialized to "closed", and the source[p872] attribute set to *source* at *element*.
    2. Set *validityResult* to the result of running check popover validity[p934] given *element*, true, and null. If this throws an exception, catch it, and set *validityResult* to that exception.

       Note

       Check popover validity[p934] is called again because firing the beforetoggle[p1574] *event could have disconnected* element *or changed its* popover[p925] *attribute.*

    3. If *validityResult* is not true:

       1. Run *cleanupSteps*.
       2. If *throwExceptions* is true and *validityResult* is a DOMException, then throw *validityResult*.
       3. Return.

    4. Request an element to be removed from the top layer given *element*.
    5. Set *element*'s implicit anchor element to null.

13. Otherwise, remove an element from the top layer immediately given *element*.
14. Set *element*'s popover trigger[p927] to null.
15. Set *element*'s opened in popover mode[p927] to null.
16. Set *element*'s popover visibility state[p926] to hidden[p926].
17. If *element* is *document*'s hint stack parent[p927], or *document*'s showing hint popover list[p935] is empty, then set *document*'s hint stack parent[p927] to null.
18. If *fireEvents* is true, then queue a popover toggle event task[p929] given *element*, "open", "closed", and *source*.
19. Let *previouslyFocusedElement* be *element*'s previously focused element[p679].
20. If *previouslyFocusedElement* is not null:

    1. Set *element*'s previously focused element[p679] to null.
    2. If *focusPreviousElement* is true and *document*'s focused area of the document[p875]'s DOM anchor[p874] is a shadow-including inclusive descendant of *element*, then run the focusing steps[p881] for *previouslyFocusedElement*; the viewport should not be scrolled by doing this step.

21. Run *cleanupSteps*.

The **togglePopover(*options*)** method steps are:

1. Let *force* be null.

2. If *options* is a boolean, set *force* to *options*.
3. Otherwise, if *options*["force"] exists, set *force* to *options*["force"].
4. Let *source* be *options*["source"] if it exists; otherwise, null.
5. If this's popover visibility state is showing, and *force* is null or false, then run the hide popover algorithm given this, true, true, true, and null.
6. Otherwise, if *force* is null or true, then run show popover given this, true, and *source*.
7. Otherwise:
    1. Let *expectedToBeShowing* be true if this's popover visibility state is showing; otherwise false.
    2. Run check popover validity given this, *expectedToBeShowing*, and null.
8. Return true if this's popover visibility state is showing; otherwise false.

To **hide popovers until**, given a Document *document*, an HTML element or null *endpoint*, a boolean *focusPreviousElement*, and a boolean *fireEvents*:

1. Let *endpointIsHint* be true if *document*'s showing hint popover list contains *endpoint*; otherwise false.
2. Run hide popover stack until given *document*, *endpoint*, Hint, *focusPreviousElement*, and *fireEvents*.
3. Let *autoEndpoint* be *endpoint*.
4. If *endpointIsHint* is true, then set *autoEndpoint* to *document*'s hint stack parent.

    Note

    This means, if endpoint *is a hint popover, auto popovers are closed, except those that are parent to that hint popover.*

5. Run hide popover stack until given *document*, *autoEndpoint*, Auto, *focusPreviousElement*, and *fireEvents*.

To **hide popover stack until**, given a Document *document*, an HTML element or null *endpoint*, an Auto or Hint *stackType*, a boolean *focusPreviousElement*, and a boolean *fireEvents*:

1. Let *popoverList* be *document*'s showing auto popover list if *stackType* is Auto; otherwise *document*'s showing hint popover list.
2. Let *lastHideIndex* be 0 if *popoverList* does not contain *endpoint*; otherwise the index of *endpoint* in *popoverList* plus 1.
3. Let *toHide* be a slice of *popoverList* from *lastHideIndex*, in reverse order.
4. Let *toRemain* be a slice of *popoverList* from 0 to *lastHideIndex*.
5. For each *popover* of *toHide*: run the hide popover algorithm given *popover*, *focusPreviousElement*, *fireEvents*, false, and null.
6. Let *newPopoverList* be *document*'s showing auto popover list if *stackType* is Auto; otherwise *document*'s showing hint popover list.
7. Let *toCheck* be *newPopoverList* in reverse order.
8. For each *popover* of *toCheck*:
    1. If *toRemain* contains *popover*, then continue.
    2. Run the hide popover algorithm given *popover*, *focusPreviousElement*, false, false, and null.

        Note

        This happens if popovers are shown whilst hiding popovers. For example, in beforetoggle *events. This is usually a developer error, so user agents are encouraged to show a warning. In this additional hiding phase,* fireEvents *is ignored, and false is used instead.*

Note

The hide popover stack until algorithm is used in several cases to hide all popovers that don't stay open when something

happens. For example, during light-dismiss of a popover, this algorithm ensures that we close only the popovers that aren't related to the node clicked by the user.

To find the **topmost popover ancestor**, given a Node *newPopoverOrTopLayerElement*, an HTML element[p46] or null *source*, and a boolean *isPopover*, perform the following steps. They return an HTML element[p46] or null.

Note

The topmost popover ancestor[p933] algorithm will return the topmost ancestor popover for the provided popover or top layer element. Popovers can be related to each other in several ways, creating a tree of popovers. There are two paths through which one popover (call it the "child" popover) can have a topmost ancestor popover (call it the "parent" popover):

1. *The popovers are nested within each other in the node tree. In this case, the descendant popover is the "child" and its topmost ancestor popover is the "parent".*
2. *An element is the 'source' of the popover (e.g., a button[p585] with command[p586] in the Show Popover[p586] state). In this case, the popover is the "child", and the popover subtree the trigger element is in is the "parent". The trigger element has to be in a popover and reference an open popover.*

In each of the relationships formed above, the parent popover has to be strictly earlier in the showing auto popover list[p934] or showing hint popover list[p935] than the child popover, or it does not form a valid ancestral relationship. This eliminates non-showing popovers and self-pointers (e.g., a popover containing an invoking element that points back to the containing popover), and it allows for the construction of a well-formed tree from the (possibly cyclic) graph of connections. Only Auto[p926] and Hint[p926] popovers are considered.

If the provided element is a top layer element such as a dialog[p675] which is not showing as a popover, then topmost popover ancestor[p933] will only look in the node tree to find the first popover.

1. If *isPopover* is true:
    1. Assert: *newPopoverOrTopLayerElement* is an HTML element[p46].
    2. Assert: *newPopoverOrTopLayerElement*'s popover[p925] attribute is not in the No Popover State[p926] or the Manual[p926] state.
    3. Assert: *newPopoverOrTopLayerElement*'s popover visibility state[p926] is not in the popover showing state[p926].
2. Otherwise:
    1. Assert: *source* is null.
3. Let *document* be *newPopoverOrTopLayerElement*'s node document.
4. Let *combinedPopovers* be *document*'s showing auto popover list[p934] extended with *document*'s showing hint popover list[p935].
5. Let *popoverAncestorIndex* be the index of the last item in *combinedPopovers* of which *newPopoverOrTopLayerElement* is a flat tree descendant, otherwise -1.
6. Let *sourceAncestorIndex* be -1.
7. If *source* is not null, then set *sourceAncestorIndex* to the index of the last item in *combinedPopovers* of which *source* is a flat tree descendant, otherwise -1.
8. Let *ancestorIndex* be the maximum of *popoverAncestorIndex* and *sourceAncestorIndex*.
9. If *ancestorIndex* is -1, then return null.
10. Return *combinedPopovers*[*ancestorIndex*].

To find the **nearest inclusive open popover** given a Node *node*, perform the following steps. They return an HTML element[p46] or null.

1. Let *currentNode* be *node*.
2. While *currentNode* is not null:

1. If *currentNode*'s opened in popover mode[p927] is "auto" or "hint", and *currentNode*'s popover visibility state[p926] is showing[p926], then return *currentNode*.
2. Set *currentNode* to *currentNode*'s parent in the flat tree.

3. Return null.

To **find the topmost auto or hint popover** given a Document[p135] *document*, perform the following steps. They return an HTML element[p46] or null.

1. If *document*'s showing hint popover list[p935] is not empty, then return *document*'s showing hint popover list[p935]'s last element.
2. If *document*'s showing auto popover list[p934] is not empty, then return *document*'s showing auto popover list[p934]'s last element.
3. Return null.

To perform the **popover focusing steps** for an HTML element[p46] *subject*:

1. If the allow focus steps[p887] given *subject*'s node document return false, then return.
2. If *subject* is a dialog[p675] element, then run the dialog focusing steps[p683] given *subject* and return.
3. If *subject* has the autofocus[p887] attribute, then let *control* be *subject*.
4. Otherwise, let *control* be the autofocus delegate[p881] for *subject* given "other".
5. If *control* is null, then return.
6. Run the focusing steps[p881] given *control*.
7. Let *topDocument* be *control*'s node navigable[p1036]'s top-level traversable[p1037]'s active document[p1036].
8. If *control*'s node document's origin is not the same[p940] as the origin of *topDocument*, then return.
9. Empty *topDocument*'s autofocus candidates[p888].
10. Set *topDocument*'s autofocus processed flag[p888] to true.

To **check popover validity** for an HTML element[p46] *element* given a boolean *expectedToBeShowing* and a Document[p135] or null *expectedDocument*, perform the following steps. They throw an exception or return a boolean.

1. If *element*'s popover[p925] attribute is in the No Popover[p926] state, then throw a "NotSupportedError" DOMException.
2. If any of the following are true:
   - *expectedToBeShowing* is true and *element*'s popover visibility state[p926] is not showing[p926]; or
   - *expectedToBeShowing* is false and *element*'s popover visibility state[p926] is not hidden[p926],

   then return false.
3. If any of the following are true:
   - *element* is not connected;
   - *element*'s node document is not fully active[p1051];
   - *expectedDocument* is not null and *element*'s node document is not *expectedDocument*;
   - *element* is a dialog[p675] element and its is modal[p680] is set to true; or
   - *element*'s fullscreen flag is set,

   then throw an "InvalidStateError" DOMException.
4. Return true.

To get the **showing auto popover list** for a Document[p135] *document*:

1. Let *popovers* be « ».

2. For each Element *element* of *document*'s top layer:

    1. If all of the following are true:

        - *element* is an HTML element[p46];
        - *element*'s opened in popover mode[p927] is "auto"; and
        - *element*'s popover visibility state[p926] is showing[p926],

        then append *element* to *popovers*.

3. Return *popovers*.

To get the **showing hint popover list** for a Document[p135] *document*:

1. Let *popovers* be « ».
2. For each Element *element* of *document*'s top layer:

    1. If all of the following are true:

        - *element* is an HTML element[p46];
        - *element*'s opened in popover mode[p927] is "hint"; and
        - *element*'s popover visibility state[p926] is showing[p926],

        then append *element* to *popovers*.

3. Return *popovers*.

6.12.1 The popover target attributes §[p935]

Buttons[p532] may have the following content attributes:

- **popovertarget**
- **popovertargetaction**

If specified, the popovertarget[p935] attribute value must be the ID of an element with a popover[p925] attribute in the same tree as the button[p532] with the popovertarget[p935] attribute.

The popovertargetaction[p935] attribute is an enumerated attribute[p80] with the following keywords and states:

Keyword	State	Brief description
toggle	**Toggle**	Shows or hides the targeted popover element.
show	**Show**	Shows the targeted popover element.
hide	**Hide**	Hides the targeted popover element.

The attribute's *missing value default*[p80] and *invalid value default*[p80] are both the Toggle[p935] state.

Note

Whenever possible ensure the popover element is placed immediately after its triggering element in the DOM. Doing so will help ensure that the popover is exposed in a logical programmatic reading order for users of assistive technology, such as screen readers.

Example

The following shows how the popovertarget[p935] attribute in combination with the popovertargetaction[p935] attribute can be used to show and close a popover:

```
<button popovertarget="foo" popovertargetaction="show">
  Show a popover
```

```
</button>

<article popover="auto" id="foo">
  This is a popover article!
  <button popovertarget="foo" popovertargetaction="hide">Close</button>
</article>
```

If a popovertargetaction[p935] attribute is not specified, the default action will be to toggle the associated popover. The following shows how only specifying the popovertarget[p935] attribute on its invoking button can toggle a manual popover between its opened and closed states. A manual popover will not respond to light dismiss[p937] or close requests[p902]:

```
<input type="button" popovertarget="foo" value="Toggle the popover">

<div popover=manual id="foo">
  This is a popover!
</div>
```

DOM interface[p154]:

IDL

```
interface mixin PopoverTargetAttributes {
  [CEReactions, Reflect="popovertarget"] attribute Element? popoverTargetElement;
  [CEReactions] attribute DOMString popoverTargetAction;
};
```

The **popoverTargetAction** IDL attribute must reflect[p108] the popovertargetaction[p935] attribute, limited to only known values[p109].

To run the **popover target attribute activation behavior** given a Node *node* and a Node *eventTarget*:

1. Let *popover* be *node*'s popover target element[p936].
2. If *popover* is null, then return.
3. If *eventTarget* is a shadow-including inclusive descendant of *popover* and *popover* is a shadow-including descendant of *node*, then return.
4. If *node*'s popovertargetaction[p935] attribute is in the show[p935] state and *popover*'s popover visibility state[p926] is showing[p926], then return.
5. If *node*'s popovertargetaction[p935] attribute is in the hide[p935] state and *popover*'s popover visibility state[p926] is hidden[p926], then return.
6. If *popover*'s popover visibility state[p926] is showing[p926], then run the hide popover algorithm[p930] given *popover*, true, true, false, and *node*.
7. Otherwise:
    1. Let *validity* be the result of running check popover validity[p934] given *popover*, false, and null. If this throws an exception, catch it, and let *validity* be false.
    2. If *validity* is true, then run show popover[p927] given *popover*, false, and *node*.

To get the **popover target element** given a Node *node*, perform the following steps. They return an HTML element[p46] or null.

1. If *node* is not a button[p532], then return null.
2. If *node* is disabled[p630], then return null.
3. If *node* has a form owner[p626] and *node* is a submit button[p532], then return null.
4. Let *popoverElement* be the result of running *node*'s get the popovertarget-associated element[p112].
5. If *popoverElement* is null, then return null.
6. If *popoverElement*'s popover[p925] attribute is in the No Popover[p926] state, then return null.

7. Return *popoverElement*.

6.12.2 Popover light dismiss

Note

"Light dismiss" means that clicking outside of a popover whose popover attribute is in the Auto or Hint state will close the popover. This is in addition to how such popovers respond to close requests.

To **light dismiss open popovers**, given a PointerEvent *event*:

1. Assert: *event*'s isTrusted attribute is true.
2. Let *target* be *event*'s target.
3. Let *document* be *target*'s node document.
4. If the result of running topmost auto or hint popover given *document* is null, then return.
5. If *event*'s type is "pointerdown": set *document*'s popover pointerdown target to the result of running topmost clicked popover given *target*.
6. If *event*'s type is "pointerup":
    1. Let *ancestor* be the result of running topmost clicked popover given *target*.
    2. Let *sameTarget* be true if *ancestor* is *document*'s popover pointerdown target.
    3. Set *document*'s popover pointerdown target to null.
    4. If *sameTarget* is false, then return.
    5. Run hide popovers until given *document*, *ancestor*, false, and true.

To find the **topmost clicked popover**, given a Node *node*:

1. Let *clickedPopover* be the result of running nearest inclusive open popover given *node*.
2. Let *targetPopover* be the result of running nearest inclusive target popover given *node*.
3. If the result of getting the popover stack position given *clickedPopover* is greater than the result of getting the popover stack position given *targetPopover*, then return *clickedPopover*.
4. Return *targetPopover*.

To **get the popover stack position**, given an HTML element *popover*:

1. Let *hintList* be *popover*'s node document's showing hint popover list.
2. Let *autoList* be *popover*'s node document's showing auto popover list.
3. If *popover* is in *hintList*, then return the index of *popover* in *hintList* + the size of *autoList* + 1.
4. If *popover* is in *autoList*, then return the index of *popover* in *autoList* + 1.
5. Return 0.

To find the **nearest inclusive target popover** given a Node *node*:

1. Let *currentNode* be *node*.
2. While *currentNode* is not null:
    1. Let *targetPopover* be *currentNode*'s target popover.
    2. If *targetPopover* is not null and *targetPopover*'s popover attribute is in the Auto state or the Hint state, and *targetPopover*'s popover visibility state is showing, then return *targetPopover*.

3. Set *currentNode* to *currentNode*'s ancestor in the flat tree.

To get the **target popover** given a Node *node*, perform the following steps. They return an HTML element[p46] or null.

1. If *node* is not a `button`[p585] element, then return *node*'s popover target element[p936].
2. If *node* is disabled[p630], then return null.
3. Let *target* be the result of running *node*'s get the `commandfor`-associated element[p112].
4. If *target* is null, then return *node*'s popover target element[p936].
5. If *node* has a form owner[p626]:
    1. If *element* is a submit button[p532], then return null.
    2. If *element*'s `type`[p586] attribute is in the Reset Button[p586] state, then return null.
    3. If *element*'s `type`[p586] attribute is in the Auto[p586] state, then return null.
6. Let *command* be *node*'s `command`[p586] attribute.
7. If *command* is not in the Toggle Popover[p586], Hide Popover[p586], or Show Popover[p586] states, then return null.
8. If *target*'s `popover`[p925] attribute is in the No Popover[p926] state, then return null.
9. Return *target*.

7 Loading web pages §[p93]

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 Supporting concepts §[p93]

7.1.1 Origins §[p93]

Origins are the fundamental currency of the web's security model. Two actors in the web platform that share an origin are assumed to trust each other and to have the same authority. Actors with differing origins are considered potentially hostile versus each other, and are isolated from each other to varying degrees.

Example

For example, if Example Bank's web site, hosted at `bank.example.com`, tries to examine the DOM of Example Charity's web site, hosted at `charity.example.org`, a `"SecurityError"` `DOMException` will be raised.

An **origin** is one of the following:

An *opaque origin*

An internal value, with no serialization it can be recreated from (it is serialized as "`null`" per serialization of an origin[p939]), for which the only meaningful operation is testing for equality.

A *tuple origin*

A tuple consisting of:

- A **scheme** (an ASCII string).
- A **host** (a host).
- A **port** (null or a 16-bit unsigned integer).
- A **domain** (null or a host). Null unless stated otherwise.

Note

Origins[p939] can be shared, e.g., among multiple `Document`[p135] objects. Furthermore, origins[p939] are generally immutable. Only the domain[p939] of a tuple origin[p939] can be changed, and only through the `document.domain`[p942] API.

An origin[p939]'s equality, given origins[p939] *A* and *B*, returns true if *A* and *B* are same origin[p940]; otherwise false.

The **effective domain** of an origin[p939] *origin* is computed as follows:

1. If *origin* is an opaque origin[p939], then return null.
2. If *origin*'s domain[p939] is non-null, then return *origin*'s domain[p939].
3. Return *origin*'s host[p939].

The **serialization of an origin** is the string obtained by applying the following algorithm to the given origin[p939] *origin*:

1. If *origin* is an opaque origin[p939], then return "`null`".
2. Otherwise, let *result* be *origin*'s scheme[p939].
3. Append "`://`" to *result*.
4. Append *origin*'s host[p939], serialized, to *result*.
5. If *origin*'s port[p939] is non-null, append a U+003A COLON character (:), and *origin*'s port[p939], serialized, to *result*.
6. Return *result*.

Example

The serialization[p939] of ("https", "xn--maraa-rta.example", null, null) is "https://xn--maraa-rta.example".

¶[p940] 0

Note

There used to also be a Unicode serialization of an origin. *However, it was never widely adopted.*

Two origins[p939], *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[p939], then return true.
2. If *A* and *B* are both tuple origins[p939], their schemes[p939] and ports[p939] are identical, and their hosts[p939] are equal, then return true.
3. Return false.

Two origins[p939], *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[p939], then return true.
2. If *A* and *B* are both tuple origins[p939]:
    1. If *A* and *B*'s schemes[p939] are identical, and their domains[p939] are identical and non-null, then return true.
    2. Otherwise, if *A* and *B* are same origin[p940] and their domains[p939] are both null, return true.
3. Return false.

Example

A	*B*	same origin[p940]	same origin-domain[p940]
("https", "example.org", null, null)	("https", "example.org", null, null)	✔	✔
("https", "example.org", 314, null)	("https", "example.org", 420, null)	✘	✘
("https", "example.org", 314, "example.org")	("https", "example.org", 420, "example.org")	✘	✔
("https", "example.org", null, null)	("https", "example.org", null, "example.org")	✔	✘
("https", "example.org", null, "example.org")	("http", "example.org", null, "example.org")	✘	✘

7.1.1.1 Sites §[p940] 0

A **scheme-and-host** is a tuple of a **scheme** (an ASCII string) and a **host** (a host).

A **site** is an opaque origin[p939] or a scheme-and-host[p940].

To **obtain a site**, given an origin *origin*, run these steps:

1. If *origin* is an opaque origin[p939], then return *origin*.
2. If *origin*'s host[p939]'s registrable domain is null, then return (*origin*'s scheme[p939], *origin*'s host[p939]).
3. Return (*origin*'s scheme[p939], *origin*'s host[p939]'s registrable domain).

Two sites[p940], *A* and *B*, are said to be **same site** if the following algorithm returns true:

1. If *A* and *B* are the same opaque origin[p939], then return true.
2. If *A* or *B* is an opaque origin[p939], then return false.
3. If *A*'s and *B*'s scheme[p940] values are different, then return false.
4. If *A*'s and *B*'s host[p940] values are not equal, then return false.
5. Return true.

The **serialization of a site** is the string obtained by applying the following algorithm to the given site[p940] *site*:

1. If *site* is an opaque origin[p939], then return "`null`".
2. Let *result* be *site*[0].
3. Append "`://`" to *result*.
4. Append *site*[1], serialized, to *result*.
5. Return *result*.

⚠Warning!

It needs to be clear from context that the serialized value is a site, not an origin, as there is not necessarily a syntactic difference between the two. For example, the origin (`"https"`, `"shop.example"`, null, null) and the site (`"https"`, `"shop.example"`) have the same serialization: `"https://shop.example"`.

Two origins[p939], *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[p939], then return true.
2. If *A* and *B* are both tuple origins[p939]:
    1. Let *hostA* be *A*'s host[p939], and let *hostB* be *B*'s host[p939].
    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[p939], *A* and *B*, are said to be **same site** if the following algorithm returns true:

1. Let *siteA* be the result of obtaining a site[p940] given *A*.
2. Let *siteB* be the result of obtaining a site[p940] given *B*.
3. If *siteA* is same site[p940] with *siteB*, then return true.
4. Return false.

Note

Unlike the same origin[p940] and same origin-domain[p940] concepts, for schemelessly same site[p941] and same site[p941], the port[p939] and domain[p939] components are ignored.

⚠Warning!

For the reasons explained in URL, the same site[p941] and schemelessly same site[p941] concepts should be avoided when possible, in favor of same origin[p940] checks.

¶[p94] Example

Given that `wildlife.museum`, `museum`, and `com` are public suffixes and that `example.com` is not:

A	*B*	schemelessly same site[p941]	same site[p941]
("https", "example.com")	("https", "sub.example.com")	✔	✔
("https", "example.com")	("https", "sub.other.example.com")	✔	✔
("https", "example.com")	("http", "non-secure.example.com")	✔	×
("https", "r.wildlife.museum")	("https", "sub.r.wildlife.museum")	✔	✔
("https", "r.wildlife.museum")	("https", "sub.other.r.wildlife.museum")	✔	✔
("https", "r.wildlife.museum")	("https", "other.wildlife.museum")	×	×
("https", "r.wildlife.museum")	("https", "wildlife.museum")	×	×
("https", "wildlife.museum")	("https", "wildlife.museum")	✔	✔
("https", "example.com")	("https", "example.com.")	×	×

(Here we have omitted the port[p939] and domain[p939] components since they are not considered.)

7.1.1.2 Relaxing the same-origin restriction §[p94]

For web developers (non-normative)

***document*.domain[p942] [= *domain*]**

Returns the current domain used for security checks.

Can be set to a value that removes subdomains, to change the origin[p939]'s domain[p939] 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 `iframe`[p404]s, `Document`[p135]s with opaque origins[p939], and `Document`[p135]s without a browsing context[p1046], the setter will throw a `"SecurityError"` exception. In cases where `crossOriginIsolated`[p1219] or `originAgentCluster`[p944] return true, the setter will do nothing.

Avoid using the `document.domain`[p942] 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`[p942] 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()`[p1296] or `MessageChannel`[p1298] objects to communicate across origins in a safe manner.

The **`domain`** getter steps are:

1. Let *effectiveDomain* be this's origin's effective domain[p939].
2. If *effectiveDomain* is null, then return the empty string.
3. Return *effectiveDomain*, serialized.

The `domain`[p942] setter steps are:

1. If this's browsing context[p1046] is null, then throw a `"SecurityError"` `DOMException`.
2. If this's active sandboxing flag set[p959] has its sandboxed `document.domain` browsing context flag[p958] set, then throw a `"SecurityError"` `DOMException`.
3. Let *effectiveDomain* be this's origin's effective domain[p939].
4. If *effectiveDomain* is null, then throw a `"SecurityError"` `DOMException`.
5. If the given value is not a registrable domain suffix of and is not equal to[p942] *effectiveDomain*, then throw a `"SecurityError"` `DOMException`.
6. If the surrounding agent's agent cluster's is origin-keyed[p1141] is true, then return.
7. Set this's origin's domain[p939] to the result of parsing the given value.

To determine if a scalar value string *hostSuffixString* **is a registrable domain suffix of or is equal to** a host *originalHost*:

1. If *hostSuffixString* is the empty string, then return false.
2. Let *hostSuffix* be the result of parsing *hostSuffixString*.
3. If *hostSuffix* is failure, then return false.
4. If *hostSuffix* does not equal *originalHost*:
    1. If *hostSuffix* or *originalHost* is not a domain, then return false.

       Note

       This excludes hosts that are IP addresses.

    2. If *hostSuffix*, prefixed by U+002E (.), does not match the end of *originalHost*, then return false.

3. If any of the following are true:

    - *hostSuffix* equals *hostSuffix*'s public suffix; or
    - *hostSuffix*, prefixed by U+002E (.), matches the end of *originalHost*'s public suffix,

    then return false. [URL]p1587

4. Assert: *originalHost*'s public suffix, prefixed by U+002E (.), matches the end of *hostSuffix*.

5. Return true.

¶p943 Example

hostSuffixString	*originalHost*	Outcome of is a registrable domain suffix of or is equal top942	Notes
"0.0.0.0"	0.0.0.0	✔	
"0x10203"	0.1.2.3	✔	
"[0::1]"	::1	✔	
"example.com"	example.com	✔	
"example.com"	example.com.	×	Trailing dot is significant.
"example.com."	example.com	×	
"example.com"	www.example.com	✔	
"com"	example.com	×	At the time of writing, com is a public suffix.
"example"	example	✔	
"compute.amazonaws.com"	example.compute.amazonaws.com	×	At the time of writing, *.compute.amazonaws.com is a public suffix.
"example.compute.amazonaws.com"	www.example.compute.amazonaws.com	×	
"amazonaws.com"	www.example.compute.amazonaws.com	×	
"amazonaws.com"	test.amazonaws.com	✔	At the time of writing, amazonaws.com is a registrable domain.

7.1.1.3 The Originp943 interface §p943

The Originp943 interface represents an originp939, allowing robust same originp940 and same sitep941 comparisons.

IDL

```
[Exposed=*]
interface Origin {
  constructor();

  static Origin from(any value);

  readonly attribute boolean opaque;

  boolean isSameOrigin(Origin other);
  boolean isSameSite(Origin other);
};
```

Originp943 objects have an associated **origin**, which holds an originp939.

Platform objects have an **extract an origin** operation, which returns null unless otherwise specified.

Objects implementing the Originp943 interface's extract an originp943 steps are to return this's originp943.

The **new Origin()** constructor steps are to set this's originp943 to a unique opaque originp939.

The static **from(*value*)** method steps are:

1. If *value* is a platform object:

    1. Let *origin* be the result of executing *value*'s extract an originp943 operation.

2. If *origin* is not null, then return a new Origin object whose origin is *origin*.

2. If *value* is a string:

    1. Let *parsedURL* be the result of basic URL parsing *value*.

    2. If *parsedURL* is not failure, then return a new Origin object whose origin is set to *parsedURL*'s origin.

3. Throw a TypeError.

The **opaque** getter steps are to return true if this's origin is an opaque origin; otherwise false.

The **isSameOrigin(*other*)** method steps are to return true if this's origin is same origin with *other*'s origin; otherwise false.

The **isSameSite(*other*)** method steps are to return true if this's origin is same site with *other*'s origin; otherwise false.

7.1.2 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 Documents (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.

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

Documents 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, Documents 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.

7.1.3 Cross-origin opener policies §[p945]

An **opener policy value** allows a document which is navigated to in a top-level browsing context[p1049] to force the creation of a new top-level browsing context[p1049], and a corresponding group[p1050]. The possible values are:

"**unsafe-none**"

This is the (current) default and means that the document will occupy the same top-level browsing context[p1049] as its predecessor, unless that document specified a different opener policy[p945].

"**same-origin-allow-popups**"

This forces the creation of a new top-level browsing context[p1049] for the document, unless its predecessor specified the same opener policy[p945] and they are same origin[p940].

"**same-origin**"

This behaves the same as "same-origin-allow-popups[p945]", with the addition that any auxiliary browsing context[p1046] created needs to contain same origin[p940] documents that also have the same opener policy[p945] or it will appear closed to the opener.

"**same-origin-plus-COEP**"

This behaves the same as "same-origin[p945]", with the addition that it sets the (new) top-level browsing context[p1049]'s group[p1050]'s cross-origin isolation mode[p1050] to one of "logical[p1050]" or "concrete[p1050]".

Note

"same-origin-plus-COEP[p945]" cannot be directly set via the `Cross-Origin-Opener-Policy[p946]` header, but results from a combination of setting both `Cross-Origin-Opener-Policy[p946]: same-origin[p945]` and a `Cross-Origin-Embedder-Policy[p955]` header whose value is compatible with cross-origin isolation[p955] together.

"**noopener-allow-popups**"

This forces the creation of a new top-level browsing context[p1049] for the document, regardless of its predecessor.

Note

While including a noopener-allow-popups[p945] value severs the opener relationship between the document on which it is applied and its opener, it does not create a robust security boundary between those same-origin documents.

Other risks from same-origin applications include:

- *Same-origin requests fetching the document's content — could be mitigated through Fetch Metadata filtering. [FETCHMETADATA][p1582]*
- *Same-origin framing - could be mitigated through X-Frame-Options[p1136] or CSP frame-ancestors.*
- *JavaScript accessible cookies - can be mitigated by ensuring all cookies are `httponly`.*
- *localStorage[p1352] access to sensitive data.*
- *Service worker installation.*
- *Cache API manipulation or access to sensitive data. [SW][p1587]*
- *`postMessage` or BroadcastChannel[p1303] messaging that exposes sensitive information.*
- *Autofill which may not require user interaction for same-origin documents.*

Developers using noopener-allow-popups[p945] need to make sure that their sensitive applications don't rely on client-side features accessible to other same-origin documents, e.g., localStorage[p1352] and other client-side storage APIs, BroadcastChannel[p1303] and related same-origin communication mechanisms. They also need to make sure that their server-side endpoints don't return sensitive data to non-navigation requests, whose response content is accessible to same-origin documents.

An **opener policy** consists of:

- A **value**, which is an opener policy value[p945], initially "unsafe-none[p945]".
- A **reporting endpoint**, which is string or null, initially null.

- A **report-only value**, which is an opener policy value[p945], initially "unsafe-none[p945]".
- A **report-only reporting endpoint**, which is a string or null, initially null.

To **match opener policy values**, given an opener policy value[p945] *documentCOOP*, an origin[p939] *documentOrigin*, an opener policy value[p945] *responseCOOP*, and an origin[p939] *responseOrigin*:

1. If *documentCOOP* is "unsafe-none[p945]" and *responseCOOP* is "unsafe-none[p945]", then return true.
2. If *documentCOOP* is "unsafe-none[p945]" or *responseCOOP* is "unsafe-none[p945]", then return false.
3. If *documentCOOP* is *responseCOOP* and *documentOrigin* is same origin[p940] with *responseOrigin*, then return true.
4. Return false.

7.1.3.1 The headers §[p94 6]

A Document[p135]'s cross-origin opener policy[p136] is derived from the `Cross-Origin-Opener-Policy` and `Cross-Origin-Opener-Policy-Report-Only` HTTP response headers. These headers are structured headers whose value must be a token. [STRUCTURED-FIELDS][p1586]

The valid token values are the opener policy values[p945]. The token may also have attached parameters; of these, the "**report-to**" parameter can have a string value that is the name of an appropriate endpoint. [REPORTING][p1584]

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 an opener policy** given a response *response* and an environment[p1143] *reservedEnvironment*:

1. Let *policy* be a new opener policy[p945].
2. If *reservedEnvironment* is a non-secure context[p1152], then return *policy*.
3. Let *parsedItem* be the result of getting a structured field value given `Cross-Origin-Opener-Policy[p946]` and "item" from *response*'s header list.
4. If *parsedItem* is not null and *parsedItem*[0] is a token:
    1. If *parsedItem*[0] is "same-origin[p945]":
        1. Let *coep* be the result of obtaining a cross-origin embedder policy[p955] from *response* and *reservedEnvironment*.
        2. If *coep*'s value[p955] is compatible with cross-origin isolation[p955], then set *policy*'s value[p945] to "same-origin-plus-COEP[p945]".
        3. Otherwise, set *policy*'s value[p945] to "same-origin[p945]".
    2. If *parsedItem*[0] is "same-origin-allow-popups[p945]", then set *policy*'s value[p945] to "same-origin-allow-popups[p945]".
    3. If *parsedItem*[0] is "noopener-allow-popups[p945]", then set *policy*'s value[p945] to "noopener-allow-popups[p945]".
    4. If *parsedItem*[1]["report-to[p946]"] exists and it is a string, then set *policy*'s reporting endpoint[p945] to *parsedItem*[1]["report-to[p946]"].
5. Set *parsedItem* to the result of getting a structured field value given `Cross-Origin-Opener-Policy-Report-Only[p946]` and "item" from *response*'s header list.
6. If *parsedItem* is not null and *parsedItem*[0] is a token:
    1. If *parsedItem*[0] is "same-origin[p945]":
        1. Let *coep* be the result of obtaining a cross-origin embedder policy[p955] from *response* and

reservedEnvironment.

2. If *coep*'s value[p955] is compatible with cross-origin isolation[p955] or *coep*'s report-only value[p955] is compatible with cross-origin isolation[p955], then set *policy*'s report-only value[p946] to "same-origin-plus-COEP[p945]".

   Note

   Report only COOP also considers report-only COEP to assign the special "same-origin-plus-COEP[p945]" value. This allows developers more freedom in the order of deployment of COOP and COEP.

3. Otherwise, set *policy*'s report-only value[p946] to "same-origin[p945]".

2. If *parsedItem*[0] is "same-origin-allow-popups[p945]", then set *policy*'s report-only value[p946] to "same-origin-allow-popups[p945]".

3. If *parsedItem*[1]["report-to[p946]"] exists and it is a string, then set *policy*'s report-only reporting endpoint[p946] to *parsedItem*[1]["report-to[p946]"].

7. Return *policy*.

7.1.3.2 Browsing context group switches due to opener policy §[p947]

To **check if popup COOP values require a browsing context group switch**, given two origins[p939] *responseOrigin* and *activeDocumentNavigationOrigin*, and two opener policy values[p945] *responseCOOPValue* and *activeDocumentCOOPValue*:

1. If *responseCOOPValue* is "noopener-allow-popups[p945]", then return true.

2. If all of the following are true:

   - *activeDocumentCOOPValue* is "same-origin-allow-popups[p945]" or "noopener-allow-popups[p945]"; and
   - *responseCOOPValue* is "unsafe-none[p945]",

   then return false.

3. If the result of matching[p946] *activeDocumentCOOPValue*, *activeDocumentNavigationOrigin*, *responseCOOPValue*, and *responseOrigin* is true, then return false.

4. Return true.

To **check if COOP values require a browsing context group switch**, given a boolean *isInitialAboutBlank*, two origins[p939] *responseOrigin* and *activeDocumentNavigationOrigin*, and two opener policy values[p945] *responseCOOPValue* and *activeDocumentCOOPValue*:

1. If *isInitialAboutBlank* is true, then return the result of checking if popup COOP values requires a browsing context group switch[p947] with *responseOrigin*, *activeDocumentNavigationOrigin*, *responseCOOPValue*, and *activeDocumentCOOPValue*.

   Note

2. *Here we are dealing with a non-popup navigation.*

   If the result of matching[p946] *activeDocumentCOOPValue*, *activeDocumentNavigationOrigin*, *responseCOOPValue*, and *responseOrigin* is true, 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[p939] *responseOrigin*, *activeDocumentNavigationOrigin*, and two opener policies[p945] *responseCOOP* and *activeDocumentCOOP*:

1. If the result of checking if COOP values require a browsing context group switch[p947] given *isInitialAboutBlank*, *responseOrigin*, *activeDocumentNavigationOrigin*, *responseCOOP*'s report-only value[p946], and *activeDocumentCOOP*'s report-only value[p946] is false, then return false.

Note

Matching report-only policies allows a website to specify the same report-only 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[p947] given *isInitialAboutBlank*, *responseOrigin*, *activeDocumentNavigationOrigin*, *responseCOOP*'s value[p945], and *activeDocumentCOOP*'s report-only value[p946] is true, then return true.
3. If the result of checking if COOP values require a browsing context group switch[p947] given *isInitialAboutBlank*, *responseOrigin*, *activeDocumentNavigationOrigin*, *responseCOOP*'s report-only value[p946], and *activeDocumentCOOP*'s value[p945] is true, then return true.
4. Return false.

An **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[p939] **origin**.
- An opener policy[p945] **opener policy**.
- A boolean **current context is navigation source**, initially false.

To **enforce a response's opener policy**, given a browsing context[p1046] *browsingContext*, a URL *responseURL*, an origin[p939] *responseOrigin*, an opener policy[p945] *responseCOOP*, an opener policy enforcement result[p948] *currentCOOPEnforcementResult*, and a referrer *referrer*:

1. Let *newCOOPEnforcementResult* be a new opener policy enforcement result[p948] with

   needs a browsing context group switch[p948]
   currentCOOPEnforcementResult's needs a browsing context group switch[p948]
   would need a browsing context group switch due to report-only[p948]
   currentCOOPEnforcementResult's would need a browsing context group switch due to report-only[p948]
   url[p948]
   responseURL
   origin[p948]
   responseOrigin
   opener policy[p948]
   responseCOOP
   current context is navigation source[p948]
   true

2. Let *isInitialAboutBlank* be *browsingContext*'s active document[p1046]'s is initial `about:blank`[p136].
3. If *isInitialAboutBlank* is true and *browsingContext*'s initial URL[p1046] is null, set *browsingContext*'s initial URL[p1046] to *responseURL*.
4. If the result of checking if COOP values require a browsing context group switch[p947] given *isInitialAboutBlank*, *responseOrigin*, *currentCOOPEnforcementResult*'s origin[p948], *responseCOOP*'s value[p945], and *currentCOOPEnforcementResult*'s opener policy[p948]'s value[p945] is true:
   1. Set *newCOOPEnforcementResult*'s needs a browsing context group switch[p948] to true.
   2. If *browsingContext*'s group[p1050]'s browsing context set[p1050]'s size is greater than 1:
      1. Queue a violation report for browsing context group switch when navigating to a COOP response[p951] with *responseCOOP*, "`enforce`", *responseURL*, *currentCOOPEnforcementResult*'s url[p948], *currentCOOPEnforcementResult*'s origin[p948], *responseOrigin*, and *referrer*.
      2. Queue a violation report for browsing context group switch when navigating away from a COOP response[p951] with *currentCOOPEnforcementResult*'s opener policy[p948], "`enforce`", *currentCOOPEnforcementResult*'s url[p948], *responseURL*, *currentCOOPEnforcementResult*'s origin[p948], *responseOrigin*, and *currentCOOPEnforcementResult*'s current context is navigation source[p948].

5. If the result of checking if enforcing report-only COOP would require a browsing context group switch[p947] given *isInitialAboutBlank*, *responseOrigin*, *currentCOOPEnforcementResult*'s origin[p948], *responseCOOP*, and *currentCOOPEnforcementResult*'s opener policy[p948] is true:

    1. Set *newCOOPEnforcementResult*'s would need a browsing context group switch due to report-only[p948] to true.

    2. If *browsingContext*'s group[p1050]'s browsing context set[p1050]'s size is greater than 1:

        1. Queue a violation report for browsing context group switch when navigating to a COOP response[p951] with *responseCOOP*, "`reporting`", *responseURL*, *currentCOOPEnforcementResult*'s url[p948], *currentCOOPEnforcementResult*'s origin[p948], *responseOrigin*, and *referrer*.

        2. Queue a violation report for browsing context group switch when navigating away from a COOP response[p951] with *currentCOOPEnforcementResult*'s opener policy[p948], "`reporting`", *currentCOOPEnforcementResult*'s url[p948], *responseURL*, *currentCOOPEnforcementResult*'s origin[p948], *responseOrigin*, and *currentCOOPEnforcementResult*'s current context is navigation source[p948].

6. Return *newCOOPEnforcementResult*.

To **obtain a browsing context to use for a navigation response**, given navigation params[p1061] *navigationParams*:

1. Let *browsingContext* be *navigationParams*'s navigable[p1061]'s active browsing context[p1036].

2. If *browsingContext* is not a top-level browsing context[p1049], then return *browsingContext*.

3. Let *coopEnforcementResult* be *navigationParams*'s COOP enforcement result[p1061].

4. Let *swapGroup* be *coopEnforcementResult*'s needs a browsing context group switch[p948].

5. Let *sourceOrigin* be *browsingContext*'s active document[p1046]'s origin.

6. Let *destinationOrigin* be *navigationParams*'s origin[p1061].

7. If *sourceOrigin* is not same site[p941] with *destinationOrigin*:

    1. If either of *sourceOrigin* or *destinationOrigin* have a scheme[p939] that is not an HTTP(S) scheme and the user agent considers it necessary for *sourceOrigin* and *destinationOrigin* to be isolated from each other (for implementation-defined reasons), optionally set *swapGroup* to true.

    Note

    For example, if a user navigates from `about:settings` to `https://example.com`, the user agent could force a swap.

    Issue #10842 tracks settling on an interoperable behavior here, instead of letting this be optional.

    2. If *navigationParams*'s user involvement[p1062] is "`browser UI`[p1062]", optionally set *swapGroup* to true.

    Issue #6356 tracks settling on an interoperable behavior here, instead of letting this be optional.

8. If *browsingContext*'s group[p1050]'s browsing context set[p1050]'s size is 1, optionally set *swapGroup* to true.

Note

Some implementations swap browsing context groups here for performance reasons.

Note

The check for other contexts that could script this one is not sufficient to prevent differences in behavior that could affect a web page. Even if there are currently no other contexts, the destination page could open a window, then if the user navigates back, the previous page could expect to be able to script the opened window. Doing a swap here would break that use case.

9. If *swapGroup* is false:

    1. If *coopEnforcementResult*'s would need a browsing context group switch due to report-only[p948] is true, set *browsingContext*'s virtual browsing context group ID[p1046] to a new unique identifier.

2. Return *browsingContext*.

10. Let *newBrowsingContext* be the first return value of creating a new top-level browsing context and document[p1048].

    Note

    In this case we are going to perform a browsing context group swap. browsingContext *will not be used by the new* Document[p135] *that we are about to create[p1106]. If it is not used by other* Document[p135]*s either (such as ones in the back/forward cache), then the user agent might destroy it[p1051] at this point.*

11. Let *navigationCOOP* be *navigationParams*'s cross-origin opener policy[p1061].

12. If *navigationCOOP*'s value[p945] is "same-origin-plus-COEP[p945]", then set *newBrowsingContext*'s group[p1050]'s cross-origin isolation mode[p1050] to either "logical[p1050]" or "concrete[p1050]". 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[p1144]. "concrete[p1050]" grants access to it and "logical[p1050]" does not.

13. Let *sandboxFlags* be a clone of *navigationParams*'s final sandboxing flag set[p1061].

14. If *sandboxFlags* is not empty:

    1. Assert: *navigationCOOP*'s value[p945] is "unsafe-none[p945]".

    2. Assert: *newBrowsingContext*'s popup sandboxing flag set[p959] is empty.

    3. Set *newBrowsingContext*'s popup sandboxing flag set[p959] to *sandboxFlags*.

15. Return *newBrowsingContext*.

7.1.3.3 Reporting §[p95]

An **accessor-accessed relationship** is an enum that describes the relationship between two browsing contexts[p1046] between which an access happened. It can take the following values:

accessor is opener

The accessor browsing context[p1046] or one of its ancestors[p1049] is the opener browsing context[p1046] of the accessed browsing context[p1046]'s top-level browsing context[p1049].

accessor is openee

The accessed browsing context[p1046] or one of its ancestors[p1049] is the opener browsing context[p1046] of the accessor browsing context[p1046]'s top-level browsing context[p1049].

none

There is no opener relationship between the accessor browsing context[p1046], the accessor browsing context[p1046], or any of their ancestors[p1049].

To **check if an access between two browsing contexts should be reported**, given two browsing contexts[p1046] *accessor* and *accessed*, a JavaScript property name *P*, and an environment settings object[p1144] *environment*:

1. If *P* is not a cross-origin accessible window property name[p963], then return.

2. Assert: *accessor*'s active document[p1046] and *accessed*'s active document[p1046] are both fully active[p1051].

3. Let *accessorTopDocument* be *accessor*'s top-level browsing context[p1049]'s active document[p1046].

4. Let *accessorInclusiveAncestorOrigins* be the list obtained by taking the origin of the active document[p1036] of each of *accessor*'s active document[p1046]'s inclusive ancestor navigables[p1041].

5. Let *accessedTopDocument* be *accessed*'s top-level browsing context[p1049]'s active document[p1046].

6. Let *accessedInclusiveAncestorOrigins* be the list obtained by taking the origin of the active document[p1036] of each of *accessed*'s active document[p1046]'s inclusive ancestor navigables[p1041].

7. If any of *accessorInclusiveAncestorOrigins* are not same origin[p940] with *accessorTopDocument*'s origin, or if any of *accessedInclusiveAncestorOrigins* are not same origin[p940] with *accessedTopDocument*'s origin, then return.

   Note

   This avoids leaking information about cross-origin iframes to a top level frame with opener policy reporting.

8. If *accessor*'s top-level browsing context[p1049]'s virtual browsing context group ID[p1046] is *accessed*'s top-level browsing context[p1049]'s virtual browsing context group ID[p1046], then return.

9. Let *accessorAccessedRelationship* be a new accessor-accessed relationship[p950] with value none[p950].

10. If *accessed*'s top-level browsing context[p1049]'s opener browsing context[p1046] is *accessor* or is an ancestor[p1049] of *accessor*, then set *accessorAccessedRelationship* to accessor is opener[p950].

11. If *accessor*'s top-level browsing context[p1049]'s opener browsing context[p1046] is *accessed* or is an ancestor[p1049] of *accessed*, then set *accessorAccessedRelationship* to accessor is openee[p950].

12. Queue violation reports for accesses[p952], given *accessorAccessedRelationship*, *accessorTopDocument*'s opener policy[p136], *accessedTopDocument*'s opener policy[p136], *accessor*'s active document[p1046]'s URL, *accessed*'s active document[p1046]'s URL, *accessor*'s top-level browsing context[p1049]'s initial URL[p1046], *accessed*'s top-level browsing context[p1049]'s initial URL[p1046], *accessor*'s active document[p1046]'s origin, *accessed*'s active document[p1046]'s origin, *accessor*'s top-level browsing context[p1049]'s opener origin at creation[p1046], *accessed*'s top-level browsing context[p1049]'s opener origin at creation[p1046], *accessorTopDocument*'s referrer[p139], *accessedTopDocument*'s referrer[p139], *P*, and *environment*.

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 an opener policy[p945] *coop*, a string *disposition*, a URL *coopURL*, a URL *previousResponseURL*, two origins[p939] *coopOrigin* and *previousResponseOrigin*, and a referrer *referrer*:

1. If *coop*'s reporting endpoint[p945] is null, return.
2. Let *coopValue* be *coop*'s value[p945].
3. If *disposition* is "`reporting`", then set *coopValue* to *coop*'s report-only value[p946].
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:

key	value
disposition	*disposition*
effectivePolicy	*coopValue*
previousResponseURL	If *coopOrigin* and *previousResponseOrigin* are same origin[p940] this is the sanitization[p951] of *previousResponseURL*, null otherwise.
referrer	*serializedReferrer*
type	"`navigation-to-response`"

7. Queue *body* as "`coop`" for *coop*'s reporting endpoint[p945] with *coopURL*.

To **queue a violation report for browsing context group switch when navigating away from a COOP response** given an opener policy[p945] *coop*, a string *disposition*, a URL *coopURL*, a URL *nextResponseURL*, two origins[p939] *coopOrigin* and *nextResponseOrigin*, and a boolean *isCOOPResponseNavigationSource*:

1. If *coop*'s reporting endpoint[p945] is null, return.
2. Let *coopValue* be *coop*'s value[p945].

3. If *disposition* is "`reporting`", then set *coopValue* to *coop*'s report-only value[p946].

4. Let *body* be a new object containing the following properties:

key	value
disposition	*disposition*
effectivePolicy	*coopValue*
nextResponseURL	If *coopOrigin* and *nextResponseOrigin* are same origin[p940] or *isCOOPResponseNavigationSource* is true, this is the sanitization[p951] of *nextResponseURL*, null otherwise.
type	"`navigation-from-response`"

5. Queue *body* as "`coop`" for *coop*'s reporting endpoint[p945] with *coopURL*.

To **queue violation reports for accesses**, given an accessor-accessed relationship[p950] *accessorAccessedRelationship*, two opener policies[p945] *accessorCOOP* and *accessedCOOP*, four URLs *accessorURL*, *accessedURL*, *accessorInitialURL*, *accessedInitialURL*, four origins[p939] *accessorOrigin*, *accessedOrigin*, *accessorCreatorOrigin* and *accessedCreatorOrigin*, two referrers[p139] *accessorReferrer* and *accessedReferrer*, a string *propertyName*, and an environment settings object[p1144] *environment*:

1. If *accessorAccessedRelationship* is accessor is opener[p950]:

    1. Queue a violation report for access to an opened window[p953], given *accessorCOOP*, *accessorURL*, *accessedURL*, *accessedInitialURL*, *accessorOrigin*, *accessedOrigin*, *accessedCreatorOrigin*, *propertyName*, and *environment*.

    2. Queue a violation report for access from the opener[p953], given *accessedCOOP*, *accessedURL*, *accessorURL*, *accessedOrigin*, *accessorOrigin*, *propertyName*, and *accessedReferrer*.

2. Otherwise, if *accessorAccessedRelationship* is accessor is openee[p950]:

    1. Queue a violation report for access to the opener[p952], given *accessorCOOP*, *accessorURL*, *accessedURL*, *accessorOrigin*, *accessedOrigin*, *propertyName*, *accessorReferrer*, and *environment*.

    2. Queue a violation report for access from an opened window[p954], given *accessedCOOP*, *accessedURL*, *accessorURL*, *accessorInitialURL*, *accessedOrigin*, *accessorOrigin*, *accessorCreatorOrigin*, and *propertyName*.

3. Otherwise:

    1. Queue a violation report for access to another window[p953], given *accessorCOOP*, *accessorURL*, *accessedURL*, *accessorOrigin*, *accessedOrigin*, *propertyName*, and *environment*.

    2. Queue a violation report for access from another window[p954], given *accessedCOOP*, *accessedURL*, *accessorURL*, *accessedOrigin*, *accessorOrigin*, and *propertyName*.

To **queue a violation report for access to the opener**, given an opener policy[p945] *coop*, two URLs *coopURL* and *openerURL*, two origins[p939] *coopOrigin* and *openerOrigin*, a string *propertyName*, a referrer *referrer*, and an environment settings object[p1144] *environment*:

1. If *coop*'s reporting endpoint[p945] is null, then return.

2. Let *sourceFile*, *lineNumber*, and *columnNumber* be the relevant script URL and problematic position which triggered this report.

3. Let *serializedReferrer* be an empty string.

4. If *referrer* is a URL, set *serializedReferrer* to the serialization of *referrer*.

5. Let *body* be a new object containing the following properties:

key	value
disposition	"`reporting`"
effectivePolicy	*coop*'s report-only value[p946]
property	*propertyName*
openerURL	If *coopOrigin* and *openerOrigin* are same origin[p940], this is the sanitization[p951] of *openerURL*, null otherwise.
referrer	*serializedReferrer*
sourceFile	*sourceFile*
lineNumber	*lineNumber*
columnNumber	*columnNumber*

key	value
type	"access-to-opener"

6. Queue *body* as "coop" for *coop*'s reporting endpoint[p945] with *coopURL* and *environment*.

To **queue a violation report for access to an opened window**, given an opener policy[p945] *coop*, three URLs *coopURL*, *openedWindowURL* and *initialWindowURL*, three origins[p939] *coopOrigin*, *openedWindowOrigin*, and *openerInitialOrigin*, a string *propertyName*, and an environment settings object[p1144] *environment*:

1. If *coop*'s reporting endpoint[p945] is null, then return.
2. Let *sourceFile*, *lineNumber*, and *columnNumber* be the relevant script URL and problematic position which triggered this report.
3. Let *body* be a new object containing the following properties:

key	value
disposition	"reporting"
effectivePolicy	*coop*'s report-only value[p946]
property	*propertyName*
openedWindowURL	If *coopOrigin* and *openedWindowOrigin* are same origin[p940], this is the sanitization[p951] of *openedWindowURL*, null otherwise.
openedWindowInitialURL	If *coopOrigin* and *openerInitialOrigin* are same origin[p940], this is the sanitization[p951] of *initialWindowURL*, null otherwise.
sourceFile	*sourceFile*
lineNumber	*lineNumber*
columnNumber	*columnNumber*
type	"access-to-opener"

4. Queue *body* as "coop" for *coop*'s reporting endpoint[p945] with *coopURL* and *environment*.

To **queue a violation report for access to another window**, given an opener policy[p945] *coop*, two URLs *coopURL* and *otherURL*, two origins[p939] *coopOrigin* and *otherOrigin*, a string *propertyName*, and an environment settings object[p1144] *environment*:

1. If *coop*'s reporting endpoint[p945] is null, then return.
2. Let *sourceFile*, *lineNumber*, and *columnNumber* be the relevant script URL and problematic position which triggered this report.
3. Let *body* be a new object containing the following properties:

key	value
disposition	"reporting"
effectivePolicy	*coop*'s report-only value[p946]
property	*propertyName*
otherURL	If *coopOrigin* and *otherOrigin* are same origin[p940], this is the sanitization[p951] of *otherURL*, null otherwise.
sourceFile	*sourceFile*
lineNumber	*lineNumber*
columnNumber	*columnNumber*
type	"access-to-opener"

4. Queue *body* as "coop" for *coop*'s reporting endpoint[p945] with *coopURL* and *environment*.

To **queue a violation report for access from the opener**, given an opener policy[p945] *coop*, two URLs *coopURL* and *openerURL*, two origins[p939] *coopOrigin* and *openerOrigin*, a string *propertyName*, and a referrer *referrer*:

1. If *coop*'s reporting endpoint[p945] 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:

key	value
disposition	"reporting"

key	value
effectivePolicy	*coop*'s report-only value[p946]
property	*propertyName*
openerURL	If *coopOrigin* and *openerOrigin* are same origin[p940], this is the sanitization[p951] of *openerURL*, null otherwise.
referrer	*serializedReferrer*
type	"access-to-opener"

5. Queue *body* as "coop" for *coop*'s reporting endpoint[p945] with *coopURL*.

To **queue a violation report for access from an opened window**, given an opener policy[p945] *coop*, three URLs *coopURL*, *openedWindowURL* and *initialWindowURL*, three origins[p939] *coopOrigin*, *openedWindowOrigin*, and *openerInitialOrigin*, and a string *propertyName*:

1. If *coop*'s reporting endpoint[p945] is null, return.
2. Let *body* be a new object containing the following properties:

key	value
disposition	"reporting"
effectivePolicy	*coop*'s report-only value[p946]
property	*propertyName*
openedWindowURL	If *coopOrigin* and *openedWindowOrigin* are same origin[p940], this is the sanitization[p951] of *openedWindowURL*, null otherwise.
openedWindowInitialURL	If *coopOrigin* and *openerInitialOrigin* are same origin[p940], this is the sanitization[p951] of *initialWindowURL*, null otherwise.
type	"access-to-opener"

3. Queue *body* as "coop" for *coop*'s reporting endpoint[p945] with *coopURL*.

To **queue a violation report for access from another window**, given an opener policy[p945] *coop*, two URLs *coopURL* and *otherURL*, two origins[p939] *coopOrigin* and *otherOrigin*, and a string *propertyName*:

1. If *coop*'s reporting endpoint[p945] is null, return.
2. Let *body* be a new object containing the following properties:

key	value
disposition	"reporting"
effectivePolicy	*coop*'s report-only value[p946]
property	*propertyName*
otherURL	If *coopOrigin* and *otherOrigin* are same origin[p940], this is the sanitization[p951] of *otherURL*, null otherwise.
type	access-to-opener

3. Queue *body* as "coop" for *coop*'s reporting endpoint[p945] with *coopURL*.

7.1.4 Cross-origin embedder policies §[p954]

An **embedder policy value** is one of three strings that controls the fetching of cross-origin resources without explicit permission from resource owners.

"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.

"credentialless"

When this value is used, fetching cross-origin no-CORS resources omits credentials. In exchange, an explicit `Cross-Origin-Resource-Policy` header is not required. Other requests sent with credentials require the server's explicit permission through the CORS protocol or the `Cross-Origin-Resource-Policy` header.

⚠Warning!

Before supporting "credentialless", implementers are strongly encouraged to support both:

- ***Private Network Access***
- ***Opaque Response Blocking***

Otherwise, it would allow attackers to leverage the client's network position to read non public resources, using the cross-origin isolated capability.

An embedder policy value is **compatible with cross-origin isolation** if it is "credentialless" or "require-corp".

An **embedder policy** consists of:

- A **value**, which is an embedder policy value, initially "unsafe-none".
- 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.

7.1.4.1 The headers §

The `Cross-Origin-Embedder-Policy` and `Cross-Origin-Embedder-Policy-Report-Only` HTTP response headers allow a server to declare an embedder policy for an environment settings object. These headers are structured headers whose values must be a token. [STRUCTURED-FIELDS]

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 string value that is the name of an appropriate endpoint. [REPORTING]

Note

The processing model fails open (by defaulting to "unsafe-none") in the presence of a header that cannot be parsed as a token. This includes inadvertent lists created by combining multiple instances of the `Cross-Origin-Embedder-Policy` header present in a given response:

`Cross-Origin-Embedder-Policy`	***Final embedder policy value***
No header delivered	"unsafe-none"
`require-corp`	"require-corp"
`unknown-value`	"unsafe-none"
`require-corp, unknown-value`	"unsafe-none"
`unknown-value, unknown-value`	"unsafe-none"
`unknown-value, require-corp`	"unsafe-none"
`require-corp, require-corp`	"unsafe-none"

(The same applies to `Cross-Origin-Embedder-Policy-Report-Only`.)

To **obtain an embedder policy** from a response *response* and an environment *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, *parsedItem*[0] is a token, and *parsedItem*[0] is compatible with cross-origin isolation:
    1. Set *policy*'s value to *parsedItem*[0].

2. If *parsedItem*[1]["`report-to`[p955]"] exists and it is a string, then set *policy*'s reporting endpoint[p955] to *parsedItem*[1]["`report-to`[p955]"].

5. Set *parsedItem* to the result of getting a structured field value with `` `Cross-Origin-Embedder-Policy-Report-Only`[p955] `` and "`item`" from *response*'s header list.

6. If *parsedItem* is non-null, *parsedItem*[0] is a token, and *parsedItem*[0] is compatible with cross-origin isolation[p955]:

    1. Set *policy*'s report-only value[p955] to *parsedItem*[0].

    2. If *parsedItem*[1]["`report-to`[p955]"] exists and it is a string, then set *policy*'s report-only reporting endpoint[p955] to *parsedItem*[1]["`report-to`[p955]"].

7. Return *policy*.

7.1.4.2 Embedder policy checks §[p956]

To **check a navigation response's adherence to its embedder policy** given a response *response*, a navigable[p1035] *navigable*, and an embedder policy[p955] *responsePolicy*:

1. If *navigable* is not a child navigable[p1039], then return true.

2. Let *parentPolicy* be *navigable*'s container document[p1038]'s policy container[p136]'s embedder policy[p960].

3. If *parentPolicy*'s report-only value[p955] is compatible with cross-origin isolation[p955] and *responsePolicy*'s value[p955] is not, then queue a cross-origin embedder policy inheritance violation[p956] with *response*, "`navigation`", *parentPolicy*'s report-only reporting endpoint[p955], "`reporting`", and *navigable*'s container document[p1038]'s relevant settings object[p1151].

4. If *parentPolicy*'s value[p955] is not compatible with cross-origin isolation[p955] or *responsePolicy*'s value[p955] is compatible with cross-origin isolation[p955], then return true.

5. Queue a cross-origin embedder policy inheritance violation[p956] with *response*, "`navigation`", *parentPolicy*'s reporting endpoint[p955], "`enforce`", and *navigable*'s container document[p1038]'s relevant settings object[p1151].

6. Return false.

To **check a global object's embedder policy** given a `WorkerGlobalScope`[p1322] *workerGlobalScope*, an environment settings object[p1144] *owner*, and a response *response*:

1. If *workerGlobalScope* is not a `DedicatedWorkerGlobalScope`[p1324] object, then return true.

2. Let *policy* be *workerGlobalScope*'s embedder policy[p1323].

3. Let *ownerPolicy* be *owner*'s policy container[p1144]'s embedder policy[p960].

4. If *ownerPolicy*'s report-only value[p955] is compatible with cross-origin isolation[p955] and *policy*'s value[p955] is not, then queue a cross-origin embedder policy inheritance violation[p956] with *response*, "`worker initialization`", *ownerPolicy*'s report-only reporting endpoint[p955], "`reporting`", and *owner*.

5. If *ownerPolicy*'s value[p955] is not compatible with cross-origin isolation[p955] or *policy*'s value[p955] is compatible with cross-origin isolation[p955], then return true.

6. Queue a cross-origin embedder policy inheritance violation[p956] with *response*, "`worker initialization`", *ownerPolicy*'s reporting endpoint[p955], "`enforce`", and *owner*.

7. Return false.

To **queue a cross-origin embedder policy inheritance violation** given a response *response*, a string *type*, a string *endpoint*, a string *disposition*, and an environment settings object[p1144] *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:

key	value
type	*type*

key	value
blockedURL	*serialized*
disposition	*disposition*

3. Queue *body* as the "coep" report type[p955] for *endpoint* on *settings*.

7.1.5 Sandboxing §[p957]

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[p1063] (or browsing contexts further nested inside it), auxiliary browsing contexts[p1046] (which are protected by the sandboxed auxiliary navigation browsing context flag[p957] defined next), and the top-level browsing context[p1049] (which is protected by the sandboxed top-level navigation without user activation browsing context flag[p957] and sandboxed top-level navigation with user activation browsing context flag[p957] defined below).

If the sandboxed auxiliary navigation browsing context flag[p957] is not set, then in certain cases the restrictions nonetheless allow popups (new top-level browsing contexts[p1049]) to be opened. These browsing contexts[p1046] always have **one permitted sandboxed navigator**, set when the browsing context is created, which allows the browsing context[p1046] that created them to actually navigate them. (Otherwise, the sandboxed navigation browsing context flag[p957] 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[p1045], e.g. using the `target`[p314] attribute or the `window.open()`[p969] method.

The *sandboxed top-level navigation without user activation browsing context flag*

This flag prevents content from navigating their top-level browsing context[p1063] and prevents content from closing their top-level browsing context[p971]. It is consulted only when the sandboxed browsing context's active window[p1046] does not have transient activation[p868].

When the sandboxed top-level navigation without user activation browsing context flag[p957] is *not* set, content can navigate its top-level browsing context[p1049], but other browsing contexts[p1046] are still protected by the sandboxed navigation browsing context flag[p957] and possibly the sandboxed auxiliary navigation browsing context flag[p957].

The *sandboxed top-level navigation with user activation browsing context flag*

This flag prevents content from navigating their top-level browsing context[p1063] and prevents content from closing their top-level browsing context[p971]. It is consulted only when the sandboxed browsing context's active window[p1046] has transient activation[p868].

As with the sandboxed top-level navigation without user activation browsing context flag[p957], this flag only affects the top-level browsing context[p1049]; if it is not set, other browsing contexts[p1046] might still be protected by other flags.

The *sandboxed origin browsing context flag*

This flag forces content into an opaque origin[p1049], thus preventing it from accessing other content from the same origin[p939].

This flag also prevents script from reading from or writing to the `document.cookie` IDL attribute[p140], and blocks access to `localStorage`[p1352].

The *sandboxed forms browsing context flag*

This flag blocks form submission[p658].

The *sandboxed pointer lock browsing context flag*

This flag disables the Pointer Lock API. [POINTERLOCK][p1585]

The *sandboxed scripts browsing context flag*

This flag blocks script execution[p1151].

The *sandboxed automatic features browsing context flag*

This flag blocks features that trigger automatically, such as automatically playing a video[p452] or automatically focusing a form control[p887].

The *sandboxed* `document.domain` *browsing context flag*

This flag prevents content from using the `document.domain`[p942] setter.

The *sandbox propagates to auxiliary browsing contexts flag*

This flag prevents content from escaping the sandbox by ensuring that any auxiliary browsing context[p1046] it creates inherits the content's active sandboxing flag set[p959].

The *sandboxed modals flag*

This flag prevents content from using any of the following features to produce modal dialogs:

- `window.alert()`[p1259]
- `window.confirm()`[p1259]
- `window.print()`[p1260]
- `window.prompt()`[p1259]
- the `beforeunload`[p1574] event

The *sandboxed orientation lock browsing context flag*

This flag disables the ability to lock the screen orientation. [SCREENORIENTATION][p1586]

The *sandboxed presentation browsing context flag*

This flag disables the Presentation API. [PRESENTATION][p1585]

The *sandboxed downloads browsing context flag*

This flag prevents content from initiating or instantiating downloads, whether through downloading hyperlinks[p322] or through navigation[p1080] that gets handled as a download[p323].

The *sandboxed custom protocols navigation browsing context flag*

This flag prevents navigations toward non fetch schemes from being handed off to external software[p1073].

When the user agent is to **parse a sandboxing directive**, given a string *input* and a sandboxing flag set[p957] *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[p957].
    - The sandboxed auxiliary navigation browsing context flag[p957], unless *tokens* contains the **`allow-popups`** keyword.
    - The sandboxed top-level navigation without user activation browsing context flag[p957], unless *tokens* contains the **`allow-top-navigation`** keyword.
    - The sandboxed top-level navigation with user activation browsing context flag[p957], unless *tokens* contains either the **`allow-top-navigation-by-user-activation`** keyword or the `allow-top-navigation`[p958] keyword.

      Note

      This means that if the `allow-top-navigation`[p958] is present, the `allow-top-navigation-by-user-activation`[p958] keyword will have no effect. For this reason, specifying both is a document conformance error.

    - The sandboxed origin browsing context flag[p957], unless the *tokens* contains the **`allow-same-origin`** keyword.

      Note

      The `allow-same-origin`[p958] 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[p957], unless *tokens* contains the **allow-forms** keyword.
- The sandboxed pointer lock browsing context flag[p957], unless *tokens* contains the **allow-pointer-lock** keyword.
- The sandboxed scripts browsing context flag[p957], unless *tokens* contains the **allow-scripts** keyword.
- The sandboxed automatic features browsing context flag[p958], unless *tokens* contains the `allow-scripts`[p959] 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[p958].
- The sandbox propagates to auxiliary browsing contexts flag[p958], unless *tokens* contains the **allow-popups-to-escape-sandbox** keyword.
- The sandboxed modals flag[p958], unless *tokens* contains the **allow-modals** keyword.
- The sandboxed orientation lock browsing context flag[p958], unless *tokens* contains the **allow-orientation-lock** keyword.
- The sandboxed presentation browsing context flag[p958], unless *tokens* contains the **allow-presentation** keyword.
- The sandboxed downloads browsing context flag[p958], unless *tokens* contains the **allow-downloads** keyword.
- The sandboxed custom protocols navigation browsing context flag[p958], unless *tokens* contains either the **allow-top-navigation-to-custom-protocols** keyword, the `allow-popups`[p958] keyword, or the `allow-top-navigation`[p958] keyword.

Every top-level browsing context[p1049] has a **popup sandboxing flag set**, which is a sandboxing flag set[p957]. When a browsing context[p1046] is created, its popup sandboxing flag set[p959] must be empty. It is populated by the rules for choosing a navigable[p1044] and the obtain a browsing context to use for a navigation response[p949] algorithm.

Every `iframe`[p404] element has an **iframe sandboxing flag set**, which is a sandboxing flag set[p957]. Which flags in an `iframe` sandboxing flag set[p959] are set at any particular time is determined by the `iframe`[p404] element's `sandbox`[p409] attribute.

Every `Document`[p135] has an **active sandboxing flag set**, which is a sandboxing flag set[p957]. When the `Document`[p135] is created, its active sandboxing flag set[p959] must be empty. It is populated by the navigation algorithm[p1063].

Every CSP list *cspList* has **CSP-derived sandboxing flags**, which is a sandboxing flag set[p957]. It is the return value of the following algorithm:

1. Let *directives* be an empty ordered set.
2. For each policy in *cspList*:
    1. If *policy*'s disposition is not "`enforce`", then continue.
    2. If *policy*'s directive set contains a directive whose name is "`sandbox`", then append that directive to *directives*.
3. If *directives* is empty, then return an empty sandboxing flag set[p957].
4. Let *directive* be *directives*[*directives*'s size − 1].
5. Return the result of parsing the sandboxing directive[p958] *directive*.

To **determine the creation sandboxing flags** for a browsing context[p1046] *browsing context*, given null or an element *embedder*, return the union of the flags that are present in the following sandboxing flag sets[p957]:

- If *embedder* is null, then the flags set on *browsing context*'s popup sandboxing flag set[p959].
- If *embedder* is an element, then the flags set on *embedder*'s `iframe` sandboxing flag set[p959].
- If *embedder* is an element, then the flags set on *embedder*'s node document's active sandboxing flag set[p959].

7.1.6 `iframe`[p404] element referrer policy §[p96]

To **determine the `iframe` element referrer policy** given an element-or-null *embedder*:

1. If *embedder* is an `iframe`[p404] element, then return *embedder*'s `referrerpolicy`[p412] attribute's state's corresponding keyword.
2. Return the empty string.

Note

This is used for allowing masking of some origins in the internal ancestor origin objects list creation steps[p137].

7.1.7 Policy containers §[p96]

A **policy container** is a struct containing policies that apply to a `Document`[p135], a `WorkerGlobalScope`[p1322], or a `WorkletGlobalScope`[p1342]. It has the following items:

- A **CSP list**, which is a CSP list. It is initially empty.
- An **embedder policy**, which is an embedder policy[p955]. It is initially a new embedder policy[p955].
- A **referrer policy**, which is a referrer policy. It is initially the default referrer policy.
- An **integrity policy**, which is an integrity policy, initially a new integrity policy.
- A **report-only integrity policy**, which is an integrity policy, initially a new integrity policy.

Move other policies into the policy container.

To **clone a policy container** given a policy container[p960] *policyContainer*:

1. Let *clone* be a new policy container[p960].
2. For each *policy* in *policyContainer*'s CSP list[p960], append a copy of *policy* into *clone*'s CSP list[p960].
3. Set *clone*'s embedder policy[p960] to a copy of *policyContainer*'s embedder policy[p960].
4. Set *clone*'s referrer policy[p960] to *policyContainer*'s referrer policy[p960].
5. Set *clone*'s integrity policy[p960] to a copy of *policyContainer*'s integrity policy[p960].
6. Return *clone*.

To determine whether a URL *url* **requires storing the policy container in history**:

1. If *url*'s scheme is "`blob`", then return false.
2. If *url* is local, then return true.
3. Return false.

To **create a policy container from a fetch response** given a response *response* and an environment[p1143]-or-null *environment*:

1. If *response*'s URL's scheme is "`blob`", then return a clone[p960] of *response*'s URL's blob URL entry's environment's policy

container[p1144].

2. Let *result* be a new policy container[p960].
3. Set *result*'s CSP list[p960] to the result of parsing a response's Content Security Policies given *response*.
4. If *environment* is non-null, then set *result*'s embedder policy[p960] to the result of obtaining an embedder policy[p955] given *response* and *environment*. Otherwise, set it to "`unsafe-none`[p954]".
5. Set *result*'s referrer policy[p960] to the result of parsing the `Referrer-Policy` header given *response*. [REFERRERPOLICY][p1585]
6. Parse Integrity-Policy headers with *response* and *result*.
7. Return *result*.

To **determine navigation params policy container** given a URL *responseURL* and four policy container[p960]-or-nulls *historyPolicyContainer*, *initiatorPolicyContainer*, *parentPolicyContainer*, and *responsePolicyContainer*:

1. If *historyPolicyContainer* is not null:
    1. Assert: *responseURL* requires storing the policy container in history[p960].
    2. Return a clone[p960] of *historyPolicyContainer*.
2. If *responseURL* is `about:srcdoc`[p100]:
    1. Assert: *parentPolicyContainer* is not null.
    2. Return a clone[p960] of *parentPolicyContainer*.
3. If *responseURL* is local and *initiatorPolicyContainer* is not null, then return a clone[p960] of *initiatorPolicyContainer*.
4. If *responsePolicyContainer* is not null, then return *responsePolicyContainer*.
5. Return a new policy container[p960].

To **initialize a worker global scope's policy container** given a `WorkerGlobalScope`[p1322] *workerGlobalScope*, a response *response*, and an environment[p1143] *environment*:

1. If *workerGlobalScope*'s url[p1323] is local but its scheme is not "`blob`":
    1. Assert: *workerGlobalScope*'s owner set[p1322]'s size is 1.
    2. Set *workerGlobalScope*'s policy container[p1323] to a clone[p960] of *workerGlobalScope*'s owner set[p1322][0]'s relevant settings object[p1151]'s policy container[p1144].
2. Otherwise, set *workerGlobalScope*'s policy container[p1323] to the result of creating a policy container from a fetch response[p960] given *response* and *environment*.

7.2 APIs related to navigation and session history §[p96]

7.2.1 Security infrastructure for `Window`[p965], `WindowProxy`[p977], and `Location`[p980] objects §[p96]

Although typically objects cannot be accessed across origins[p939], 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.

This section uses the terminology and typographic conventions from the JavaScript specification. [JAVASCRIPT][p1583]

7.2.1.1 Integration with IDL §[p96]

When perform a security check is invoked, with a *platformObject*, *identifier*, and *type*, run these steps:

1. If *platformObject* is not a `Window`[p965] or `Location`[p980] object, then return.

2. For each *e* of CrossOriginProperties[p962](*platformObject*):

    1. If SameValue(*e*.[[Property]], *identifier*) is true:

        1. If *type* is "`method`" and *e* has neither [[NeedsGetter]] nor [[NeedsSetter]], then return.

        2. Otherwise, if *type* is "`getter`" and *e*.[[NeedsGetter]] is true, then return.

        3. Otherwise, if *type* is "`setter`" and *e*.[[NeedsSetter]] is true, then return.

3. If IsPlatformObjectSameOrigin[p963](*platformObject*) is false, then throw a "`SecurityError`" `DOMException`.

7.2.1.2 Shared internal slot: [[CrossOriginPropertyDescriptorMap]] §[p962]

`Window`[p965] and `Location`[p980] objects both have a **[[CrossOriginPropertyDescriptorMap]]** internal slot, whose value is initially an empty map.

The [[CrossOriginPropertyDescriptorMap]][p962] 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`[p965] or `Location`[p980] object from *objectGlobal*. It is filled lazily by CrossOriginGetOwnPropertyHelper[p963], 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`[p942] is set. This is not observable as `document.domain`[p942] cannot revisit an earlier value.

Example

For example, setting `document.domain`[p942] 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[p939] again and therefore the corresponding value could never be retrieved from the map.

7.2.1.3 Shared abstract operations §[p962]

7.2.1.3.1 CrossOriginProperties (O) §[p962]

1. Assert: *O* is a `Location`[p980] or `Window`[p965] object.

2. If *O* is a `Location`[p980] object, then return « { [[Property]]: "`href`", [[NeedsGetter]]: false, [[NeedsSetter]]: true }, { [[Property]]: "`replace`" } ».

3. Return « { [[Property]]: "`window`", [[NeedsGetter]]: true, [[NeedsSetter]]: false }, { [[Property]]: "`self`", [[NeedsGetter]]: true, [[NeedsSetter]]: false }, { [[Property]]: "`location`", [[NeedsGetter]]: true, [[NeedsSetter]]: true }, { [[Property]]: "`close`" }, { [[Property]]: "`closed`", [[NeedsGetter]]: true, [[NeedsSetter]]: false }, { [[Property]]: "`focus`" }, { [[Property]]: "`blur`" }, { [[Property]]: "`frames`", [[NeedsGetter]]: true, [[NeedsSetter]]: false }, { [[Property]]: "`length`", [[NeedsGetter]]: true, [[NeedsSetter]]: false }, { [[Property]]: "`top`", [[NeedsGetter]]: true, [[NeedsSetter]]: false }, { [[Property]]: "`opener`", [[NeedsGetter]]: true, [[NeedsSetter]]: false }, { [[Property]]: "`parent`", [[NeedsGetter]]: true, [[NeedsSetter]]: false }, { [[Property]]: "`postMessage`" } ».

Note

This abstract operation does not return a Completion Record.

Note

Indexed properties do not need to be safelisted in this algorithm, as they are handled directly by the WindowProxy[p977] 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.

7.2.1.3.2 CrossOriginPropertyFallback (P) §[p96]

1. If *P* is "then", %Symbol.toStringTag%[p58], %Symbol.hasInstance%[p58], or %Symbol.isConcatSpreadable%[p58], then return PropertyDescriptor { [[Value]]: undefined, [[Writable]]: false, [[Enumerable]]: false, [[Configurable]]: true }.

2. Throw a "SecurityError" DOMException.

7.2.1.3.3 IsPlatformObjectSameOrigin (O) §[p96]

1. Return true if the current settings object[p1151]'s origin[p1144] is same origin-domain[p940] with *O*'s relevant settings object[p1151]'s origin[p1144], and false otherwise.

Note

This abstract operation does not return a Completion Record.

Note

Here the current settings object[p1151] 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`[p966] *getter is reached as part of the [[Get]][p978] algorithm for the* `WindowProxy`[p977] *w.*

7.2.1.3.4 CrossOriginGetOwnPropertyHelper (O, P) §[p96]

Note

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[p963].

1. Let *crossOriginKey* be a tuple consisting of the current settings object[p1151], *O*'s relevant settings object[p1151], and *P*.

2. For each *e* of CrossOriginProperties[p962](*O*):

    1. If SameValue(*e*.[[Property]], *P*) is true:

        1. If the value of the [[CrossOriginPropertyDescriptorMap]][p962] internal slot of *O* contains an entry whose key equals *crossOriginKey*, then return that entry's value.

        2. Let *originalDesc* be OrdinaryGetOwnProperty(*O*, *P*).

        3. Let *crossOriginDesc* be undefined.

        4. If *e*.[[NeedsGetter]] and *e*.[[NeedsSetter]] are absent:

            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, that performs the same steps as the IDL operation *P* on object *O*.

            3. Set *crossOriginDesc* to PropertyDescriptor { [[Value]]: *value*, [[Enumerable]]: false, [[Writable]]: false, [[Configurable]]: true }.

        5. Otherwise:

1. Let *crossOriginGetter* be undefined.
2. If *e*.[[NeedsGetter]] is true, then set *crossOriginGetter* to an anonymous built-in function, created in the current realm, that performs the same steps as the getter of the IDL attribute *P* on object *O*.
3. Let *crossOriginSetter* be undefined.
4. If *e*.[[NeedsSetter]] is true, then set *crossOriginSetter* to an anonymous built-in function, created in the current realm, that performs the same steps as the setter of the IDL attribute *P* on object *O*.
5. Set *crossOriginDesc* to PropertyDescriptor { [[Getter]]: *crossOriginGetter*, [[Setter]]: *crossOriginSetter*, [[Enumerable]]: false, [[Configurable]]: true }.

6. Create an entry in the value of the [[CrossOriginPropertyDescriptorMap]][p962] internal slot of *O* with key *crossOriginKey* and value *crossOriginDesc*.
7. Return *crossOriginDesc*.

3. Return undefined.

Note

This abstract operation does not return a Completion Record.

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.) [JAVASCRIPT][p1583]

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.

7.2.1.3.5 CrossOriginGet (O, P, Receiver) §[p964]

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*.[[Getter]].
6. If *getter* is undefined, then throw a `"SecurityError"` `DOMException`.
7. Return ? Call(*getter*, *Receiver*).

7.2.1.3.6 CrossOriginSet (O, P, V, Receiver) §[p964]

1. Let *desc* be ? *O*.[[GetOwnProperty]](*P*).
2. Assert: *desc* is not undefined.
3. If *desc*.[[Setter]] is present and its value is not undefined:

1. Perform ? Call(*desc*.[[Setter]], *Receiver*, « *V* »).
2. Return true.

4. Throw a "SecurityError" DOMException.

7.2.1.3.7 CrossOriginOwnPropertyKeys (O) §p965

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", %Symbol.toStringTag%, %Symbol.hasInstance%, %Symbol.isConcatSpreadable% ».

Note

This abstract operation does not return a Completion Record.

7.2.2 The Window object §p965

IDL

```
[Global=Window,
 Exposed=Window,
 LegacyUnenumerableNamedProperties]
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;
  [Replaceable] readonly attribute Navigation navigation;
  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 = "");

  // 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.
  getter object (DOMString name);

  // the user agent
  readonly attribute Navigator navigator;
  [Replaceable] readonly attribute Navigator clientInformation; // legacy alias of .navigator
  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 : StructuredSerializeOptions {
  USVString targetOrigin = "/";
};
```

For web developers (non-normative)

***window*.window** p966

***window*.frames** p966

***window*.self** p966

These attributes all return *window*.

***window*.document** p966

Returns the Document p135 associated with *window*.

***document*.defaultView** p966

Returns the Window p965 associated with *document*, if there is one, or null otherwise.

The Window p965 object has an **associated Document**, which is a Document p135 object. It is set when the Window p965 object is created, and only ever changed during navigation p1063 from the initial about:blank p136 Document p135.

A Window p965's **browsing context** is its associated Document p966's browsing context p1046. Note *It is either null or a browsing context p1046.*

A Window p965's **navigable** is the navigable p1035 whose active document p1036 is the Window p965's associated Document p966's, or null if there is no such navigable p1035.

The **window**, **frames**, and **self** getter steps are to return this's relevant realm p1151.[[GlobalEnv]].[[GlobalThisValue]].

The **document** getter steps are to return this's associated Document p966.

Note

The Document p135 object associated with a Window p965 object can change in exactly one case: when the navigate p1063 algorithm creates a new Document object p1106 for the first page loaded in a browsing context p1046. In that specific case, the Window p965 object of the initial about:blank p136 page is reused and gets a new Document p135 object.

The **defaultView** getter steps are:

1. If this's browsing context[p1046] is null, then return null.
2. Return this's browsing context[p1046]'s WindowProxy[p977] object.

For historical reasons, Window[p965] objects must also have a writable, configurable, non-enumerable property named **HTMLDocument** whose value is the Document[p135] interface object.

7.2.2.1 Opening and closing windows §[p967]

For web developers (non-normative)

***window* = *window*.open[p969]([*url* [, *target* [, *features*]]])**

Opens a window to show *url* (defaults to "about:blank[p54]"), and returns it. *target* (defaults to "_blank") gives the name of the new window. If a window already exists with that name, it is reused. The *features* argument can contain a set of comma-separated tokens[p99]:

"noopener"

"noreferrer"

These behave equivalently to the noopener[p337] and noreferrer[p338] link types on hyperlinks[p313].

"popup"

Encourages user agents to provide a minimal web browser user interface for the new window. (Impacts the visible[p975] getter on all BarProp[p975] objects as well.)

Example

```
globalThis.open("https://email.example/message/
CAOOOkFcWW97r8yg=SsWg7GgCmp4suVX9o85y8BvNRqMjuc5PXg", undefined, "noopener,popup");
```

***window*.name[p970] [= *value*]**

Returns the name of the window.

Can be set, to change the name.

***window*.close[p971]()**

Closes the window.

***window*.closed[p971]**

Returns true if the window has been closed, false otherwise.

***window*.stop[p971]()**

Cancels the document load.

To **get noopener for window open**, given a Document[p135] *sourceDocument*, an ordered map *tokenizedFeatures*, and a URL record-or-null *url*, perform the following steps. They return a boolean.

1. If *url* is not null and *url*'s blob URL entry is not null:
    1. Let *blobOrigin* be *url*'s blob URL entry's environment's origin[p1144].
    2. Let *topLevelOrigin* be *sourceDocument*'s relevant settings object[p1151]'s top-level origin[p1144].
    3. If *blobOrigin* is not same site[p941] with *topLevelOrigin*, then return true.
2. Let *noopener* be false.
3. If *tokenizedFeatures*["noopener"] exists, then set *noopener* to the result of parsing *tokenizedFeatures*["noopener"] as a boolean feature[p970].
4. Return *noopener*.

The **window open steps**, given a string *url*, a string *target*, and a string *features*, are as follows:

1. If the event loop[p1193]'s termination nesting level[p1114] is nonzero, then return null.

2. Let *sourceDocument* be the entry global object[p1148]'s associated `Document`[p966].
3. Let *urlRecord* be null.
4. If *url* is not the empty string:
    1. Set *urlRecord* to the result of encoding-parsing a URL[p101] given *url*, relative to *sourceDocument*.
    2. If *urlRecord* is failure, then throw a `"SyntaxError"` `DOMException`.
5. If *target* is the empty string, then set *target* to "`_blank`".
6. Let *tokenizedFeatures* be the result of tokenizing[p969] *features*.
7. Let *noreferrer* be false.
8. If *tokenizedFeatures*["`noreferrer`"] exists, then set *noreferrer* to the result of parsing *tokenizedFeatures*["`noreferrer`"] as a boolean feature[p970].
9. Let *noopener* be the result of getting noopener for window open[p967] with *sourceDocument*, *tokenizedFeatures*, and *urlRecord*.
10. Remove *tokenizedFeatures*["`noopener`"] and *tokenizedFeatures*["`noreferrer`"].
11. Let *referrerPolicy* be the empty string.
12. If *noreferrer* is true, then set *noopener* to true and set *referrerPolicy* to "`no-referrer`".
13. Let *targetNavigable* and *windowType* be the result of applying the rules for choosing a navigable[p1044] given *target*, *sourceDocument*'s node navigable[p1036], and *noopener*.

    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`[p1213] handler uses the `window.open()`[p969] API to open a page in an `iframe`[p404] element, the user agent could override the selection of the target browsing context to instead target a new tab.

14. If *targetNavigable* is null, then return null.
15. If *windowType* is either "`new and unrestricted`" or "`new with no opener`":
    1. Set *targetNavigable*'s active browsing context[p1036]'s is popup[p1046] to the result of checking if a popup window is requested[p969], given *tokenizedFeatures*.
    2. Set up browsing context features for *targetNavigable*'s active browsing context[p1036] given *tokenizedFeatures*. [CSSOMVIEW][p1581]
    3. If *urlRecord* is null, then set *urlRecord* to a URL record representing `about:blank`[p54].
    4. If *urlRecord* matches `about:blank`[p100], then perform the URL and history update steps[p1078] given *targetNavigable*'s active document[p1036] and *urlRecord*.

        Note

        This is necessary in case url *is something like* `about:blank?foo`*. If* url *is just plain* `about:blank`*, this will do nothing.*

    5. Otherwise, navigate[p1063] *targetNavigable* to *urlRecord* using *sourceDocument*, with *referrerPolicy*[p1063] set to *referrerPolicy* and *exceptionsEnabled*[p1063] set to true.
16. Otherwise:
    1. If *urlRecord* is not null, then navigate[p1063] *targetNavigable* to *urlRecord* using *sourceDocument*, with *referrerPolicy*[p1063] set to *referrerPolicy* and *exceptionsEnabled*[p1063] set to true.
    2. If *noopener* is false, then set *targetNavigable*'s active browsing context[p1036]'s opener browsing context[p1046] to *sourceDocument*'s browsing context[p1046].
17. If *windowType* is "`new with no opener`", then return null.
18. If *noopener* is true and *target* is not an ASCII case-insensitive match for "`_self`", "`_parent`", or "`_top`", then return null.
19. Return *targetNavigable*'s active `WindowProxy`[p1036].

The **`open(url, target, features)`** method steps are to run the window open steps[p967] with *url*, *target*, and *features*.

Note

The method provides a mechanism for navigating[p1063] an existing browsing context[p1046] or opening and navigating an auxiliary browsing context[p1046].

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[p970] from *features* given *position*. This skips past leading separators before the name.
    4. Collect a sequence of code points that are not feature separators[p970] 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[p970] *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[p970], 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[p970]:
        1. While *position* is not past the end of *features* and the code point at *position* in *features* is a feature separator[p970]:
            1. If the code point at *position* in *features* is U+002C (,), then break.
            2. Advance *position* by 1.

            Note

            This skips to the first non-separator but does not skip past a U+002C (,).
        2. Collect a sequence of code points that are not feature separators[p970] 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*.

To **check if a window feature is set**, given *tokenizedFeatures*, *featureName*, and *defaultValue*:

1. If *tokenizedFeatures*[*featureName*] exists, then return the result of parsing *tokenizedFeatures*[*featureName*] as a boolean feature[p970].
2. Return *defaultValue*.

To **check if a popup window is requested**, given *tokenizedFeatures*:

1. If *tokenizedFeatures* is empty, then return false.
2. If *tokenizedFeatures*["popup"] exists, then return the result of parsing *tokenizedFeatures*["popup"] as a boolean feature[p970].

3. Let *location* be the result of checking if a window feature is set[p969], given *tokenizedFeatures*, "`location`", and false.
4. Let *toolbar* be the result of checking if a window feature is set[p969], given *tokenizedFeatures*, "`toolbar`", and false.
5. If *location* and *toolbar* are both false, then return true.
6. Let *menubar* be the result of checking if a window feature is set[p969], given *tokenizedFeatures*, "`menubar`", and false.
7. If *menubar* is false, then return true.
8. Let *resizable* be the result of checking if a window feature is set[p969], given *tokenizedFeatures*, "`resizable`", and true.
9. If *resizable* is false, then return true.
10. Let *scrollbars* be the result of checking if a window feature is set[p969], given *tokenizedFeatures*, "`scrollbars`", and false.
11. If *scrollbars* is false, then return true.
12. Let *status* be the result of checking if a window feature is set[p969], given *tokenizedFeatures*, "`status`", and false.
13. If *status* is false, then return true.
14. Return false.

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. If *value* is "`true`", then return true.
4. Let *parsed* be the result of parsing *value* as an integer[p80].
5. If *parsed* is an error, then set it to 0.
6. Return false if *parsed* is 0, and true otherwise.

The **`name`** getter steps are:

1. If this's navigable[p966] is null, then return the empty string.
2. Return this's navigable[p966]'s target name[p1036].

The `name`[p970] setter steps are:

1. If this's navigable[p966] is null, then return.
2. Set this's navigable[p966]'s active session history entry[p1035]'s document state[p1053]'s navigable target name[p1055] to the given value.

Note

The name gets reset[p1067] when the navigable is navigated[p1063] to another origin[p939].

The **close()** method steps are:

1. Let *thisTraversable* be this's navigable[p966].
2. If *thisTraversable* is not a top-level traversable[p1037], then return.
3. If *thisTraversable*'s is closing[p1036] is true, then return.
4. Let *browsingContext* be *thisTraversable*'s active browsing context[p1036].
5. Let *sourceSnapshotParams* be the result of snapshotting source snapshot params[p1060] given *thisTraversable*'s active document[p1036].
6. If all the following are true:
   - *thisTraversable* is script-closable[p971];
   - the incumbent global object[p1149]'s browsing context[p966] is familiar with[p1050] *browsingContext*; and
   - the incumbent global object[p1149]'s navigable[p966] is allowed by sandboxing to navigate[p1074] *thisTraversable*, given *sourceSnapshotParams*,

   then:
   1. Set *thisTraversable*'s is closing[p1036] to true.
   2. Queue a task[p1194] on the DOM manipulation task source[p1203] to definitely close[p1042] *thisTraversable*.

A navigable[p1035] is **script-closable** if it is a top-level traversable[p1037], and any of the following are true:

- its is created by web content[p1037] is true; or
- its session history entries[p1037]'s size is 1.

The **closed** getter steps are to return true if this's navigable[p966] is null or its is closing[p1036] is true; otherwise false.

The **stop()** method steps are:

1. If this's navigable[p966] is null, then return.
2. Stop loading[p1118] this's navigable[p966].

7.2.2.2 Indexed access on the Window[p965] object §[p97] 1

For web developers (non-normative)

***window*.length**[p971]

Returns the number of document-tree child navigables[p1041].

***window*[*index*]**

Returns the WindowProxy[p977] corresponding to the indicated document-tree child navigables[p1041].

The **length** getter steps are to return this's associated Document[p966]'s document-tree child navigables[p1041]'s size.

Note

Indexed access to document-tree child navigables[p1041] is defined through the [[GetOwnProperty]][p977] internal method of the WindowProxy[p977] object.

7.2.2.3 Named access on the Window[p965] object §[p97]

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 navigable target name property set** of a Window[p965] object *window* is the return value of running these steps:

1. Let *children* be the document-tree child navigables[p1041] of *window*'s associated Document[p966].
2. Let *firstNamedChildren* be an empty ordered set.
3. For each *navigable* of *children*:
    1. Let *name* be *navigable*'s target name[p1036].
    2. If *name* is the empty string, then continue.
    3. If *firstNamedChildren* contains a navigable[p1035] whose target name[p1036] is *name*, then continue.
    4. Append *navigable* to *firstNamedChildren*.
4. Let *names* be an empty ordered set.
5. For each *navigable* of *firstNamedChildren*:
    1. Let *name* be *navigable*'s target name[p1036].
    2. If *navigable*'s active document[p1036]'s origin is same origin[p940] with *window*'s relevant settings object[p1151]'s origin[p1144], then append *name* to *names*.
6. Return *names*.

Example

The two separate iterations mean that in the following example, hosted on `https://example.org/`, assuming `https://elsewhere.example/` sets `window.name`[p970] to "`spices`", evaluating `window.spices` after everything has loaded will yield undefined:

```
<iframe src=https://elsewhere.example.com/></iframe>
<iframe name=spices></iframe>
```

The Window[p965] object supports named properties. The supported property names of a Window[p965] 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 navigable target name property set[p972];
- the value of the `name` content attribute for all `embed`[p413], `form`[p532], `img`[p359], and `object`[p416] elements that have a non-empty `name` content attribute and are in a document tree with *window*'s associated Document[p966] as their root; and
- the ID of all elements that have an ID and are in a document tree with *window*'s associated Document[p966] as their root.

To determine the value of a named property *name* in a Window[p965] object *window*, the user agent must return the value obtained using the following steps:

1. Let *objects* be the list of named objects[p973] of *window* with the name *name*.

    Note

    There will be at least one such object, since the algorithm would otherwise not have been invoked by Web IDL.

2. If *objects* contains a navigable[p1035]:

1. Let *container* be the first navigable container[p1038] in *window*'s associated Document[p966]'s descendants whose content navigable[p1038] is in *objects*.

2. Return *container*'s content navigable[p1038]'s active WindowProxy[p1036].

3. Otherwise, if *objects* has only one element, return that element.

4. Otherwise, return an HTMLCollection rooted at *window*'s associated Document[p966], whose filter matches only named objects[p973] of *window* with the name *name*. (By definition, these will all be elements.)

Named objects of Window[p965] object *window* with the name *name*, for the purposes of the above algorithm, consist of the following:

- document-tree child navigables[p1041] of *window*'s associated Document[p966] whose target name[p1036] is *name*;
- embed[p413], form[p532], img[p359], or object[p416] elements that have a `name` content attribute whose value is *name* and are in a document tree with *window*'s associated Document[p966] as their root; and
- elements whose ID is *name* and that are in a document tree with *window*'s associated Document[p966] as their root.

Note

Since the Window[p965] interface has the [Global] extended attribute, its named properties follow the rules for named properties objects rather than legacy platform objects.

7.2.2.4 Accessing related windows §[p973]

For web developers (non-normative)

***window*.top**[p973]

Returns the WindowProxy[p977] for the top-level traversable[p1037].

***window*.opener**[p973] **[= *value*]**

Returns the WindowProxy[p977] for the opener browsing context[p1046].

Returns null if there isn't one or if it has been set to null.

Can be set to null.

***window*.parent**[p974]

Returns the WindowProxy[p977] for the parent navigable[p1035].

***window*.frameElement**[p974]

Returns the navigable container[p1038] element.

Returns null if there isn't one, and in cross-origin situations.

The **top** getter steps are:

1. If this's navigable[p966] is null, then return null.
2. Return this's navigable[p966]'s top-level traversable[p1037]'s active WindowProxy[p1036].

The **opener** getter steps are:

1. Let *current* be this's browsing context[p966].
2. If *current* is null, then return null.
3. If *current*'s opener browsing context[p1046] is null, then return null.
4. Return *current*'s opener browsing context[p1046]'s WindowProxy[p977] object.

The opener[p973] setter steps are:

1. If the given value is null and this's browsing context[p966] is non-null, then set this's browsing context[p966]'s opener browsing context[p1046] to null.
2. If the given value is non-null, then perform ? DefinePropertyOrThrow(this, "opener", { [[Value]]: the given value, [[Writable]]:

true, [[Enumerable]]: true, [[Configurable]]: true }).

Note

Setting `window.opener`[p973] to null clears the opener browsing context[p1046] reference. In practice, this prevents future scripts from accessing their opener browsing context[p1046]'s `Window`[p965] object.

By default, scripts can access their opener browsing context[p1046]'s `Window`[p965] object through the `window.opener`[p973] getter. E.g., a script can set `window.opener.location`, causing the opener browsing context[p1046] to navigate.

The **`parent`** getter steps are:

1. Let *navigable* be this's navigable[p966].
2. If *navigable* is null, then return null.
3. If *navigable*'s parent[p1035] is not null, then set *navigable* to *navigable*'s parent[p1035].
4. Return *navigable*'s active `WindowProxy`[p1036].

The **`frameElement`** getter steps are:

1. Let *current* be this's node navigable[p1036].
2. If *current* is null, then return null.
3. Let *container* be *current*'s container[p1038].
4. If *container* is null, then return null.
5. If *container*'s node document's origin is not same origin-domain[p940] with the current settings object[p1151]'s origin[p1144], then return null.
6. Return *container*.

Example

An example of when these properties 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[p1046] corresponding to `iframeWindow` was nulled out[p1116] when `element` was removed from the document.

7.2.2.5 Historical browser interface element APIs §[p974]

For historical reasons, the `Window`[p965] interface had some properties that represented the visibility of certain web browser interface elements.

For privacy and interoperability reasons, those properties now return values that represent whether the `Window`[p965]'s browsing context[p966]'s is popup[p1046] property is true or false.

Each interface element is represented by a BarProp[p975] object:

```
IDL
[Exposed=Window]
interface BarProp {
  readonly attribute boolean visible;
};
```

For web developers (non-normative)

***window*.locationbar[p975].visible[p975]**

***window*.menubar[p975].visible[p975]**

***window*.personalbar[p975].visible[p975]**

***window*.scrollbars[p975].visible[p975]**

***window*.statusbar[p975].visible[p975]**

***window*.toolbar[p975].visible[p975]**

Returns true if the Window[p965] is not a popup; otherwise, returns false.

The **visible** getter steps are:

1. Let *browsingContext* be this's relevant global object[p1151]'s browsing context[p966].
2. If *browsingContext* is null, then return true.
3. Return the negation of *browsingContext*'s top-level browsing context[p1049]'s is popup[p1046].

The following BarProp[p975] objects must exist for each Window[p965] object:

The location bar* BarProp *object

Historically represented the user interface element that contains a control that displays the browser's location bar.

The menu bar* BarProp *object

Historically represented the user interface element that contains a list of commands in menu form, or some similar interface concept.

The personal bar* BarProp *object

Historically represented the user interface element that contains links to the user's favorite pages, or some similar interface concept.

The scrollbar* BarProp *object

Historically represented the user interface element that contains a scrolling mechanism, or some similar interface concept.

The status bar* BarProp *object

Historically represented 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 currently indicating.

The toolbar* BarProp *object

Historically represented the user interface element found immediately above or before the document, as appropriate for the user's media, which typically provides session history traversal[p1090] controls (back and forward buttons, reload buttons, etc.).

The **locationbar** attribute must return the location bar BarProp object[p975].

The **menubar** attribute must return the menu bar BarProp object[p975].

The **personalbar** attribute must return the personal bar BarProp object[p975].

The **scrollbars** attribute must return the scrollbar BarProp object[p975].

The **statusbar** attribute must return the status bar BarProp object[p975].

The **toolbar** attribute must return the toolbar BarProp object[p975].

For historical reasons, the **status** attribute on the Window[p965] 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`[p965] object is created, the attribute must be set to the empty string. It does not do anything else.

7.2.2.6 Script settings for `Window`[p965] objects §[p97 6]

To **set up a window environment settings object**, given a URL *creationURL*, a JavaScript execution context *execution context*, null or an environment[p1143] *reservedEnvironment*, a URL *topLevelCreationURL*, and an origin[p939] *topLevelOrigin*, run these steps:

1. Let *realm* be the value of *execution context*'s Realm component.
2. Let *window* be *realm*'s global object[p1145].
3. Let *settings object* be a new environment settings object[p1144] whose algorithms are defined as follows:

   The realm execution context[p1144]
   Return *execution context*.

   The module map[p1144]
   Return the module map[p136] of *window*'s associated `Document`[p966].

   The API base URL[p1144]
   Return the current base URL[p101] of *window*'s associated `Document`[p966].

   The origin[p1144]
   Return the origin of *window*'s associated `Document`[p966].

   The has cross-site ancestor[p1144]

   1. If *window*'s navigable[p1035]'s parent[p1035] is null, then return false.
   2. Let *parentDocument* be *window*'s navigable[p1035]'s parent[p1035]'s active document[p1036].
   3. If *parentDocument*'s relevant settings object[p1151]'s has cross-site ancestor[p1144] is true, then return true.
   4. If *parentDocument*'s origin is not same site[p941] with *window*'s associated `Document`[p966]'s origin, then return true.
   5. Return false.

   The policy container[p1144]
   Return the policy container[p136] of *window*'s associated `Document`[p966].

   The cross-origin isolated capability[p1144]
   Return true if both of the following hold, and false otherwise:

   - *realm*'s agent cluster's cross-origin-isolation mode[p1141] is "`concrete`[p1050]", and
   - *window*'s associated `Document`[p966] is allowed to use[p412] the "`cross-origin-isolated`[p79]" feature.

   The time origin[p1145]
   Return *window*'s associated `Document`[p966]'s load timing info[p141]'s navigation start time[p142].

4. If *reservedEnvironment* is non-null:

   1. Set *settings object*'s id[p1143] to *reservedEnvironment*'s id[p1143], target browsing context[p1144] to *reservedEnvironment*'s target browsing context[p1144], and active service worker[p1144] to *reservedEnvironment*'s active service worker[p1144].
   2. Set *reservedEnvironment*'s id[p1143] to the empty string.

   > **Note**
   > *The identity of the reserved environment is considered to be fully transferred to the created environment settings object[p1144]. The reserved environment is not searchable by the environment[p1143]'s id[p1143] from this point on.*

5. Otherwise, set *settings object*'s id[p1143] to a new unique opaque string, *settings object*'s target browsing context[p1144] to null,

and *settings object*'s active service worker[p1144] to null.

6. Set *settings object*'s creation URL[p1143] to *creationURL*, *settings object*'s top-level creation URL[p1144] to *topLevelCreationURL*, and *settings object*'s top-level origin[p1144] to *topLevelOrigin*.

7. Set *realm*'s [[HostDefined]] field to *settings object*.

7.2.3 The `WindowProxy`[p977] exotic object §[p97 7]

A **`WindowProxy`** is an exotic object that wraps a `Window`[p965] ordinary object, indirecting most operations through to the wrapped object. Each browsing context[p1046] has an associated `WindowProxy`[p977] object. When the browsing context[p1046] is navigated[p1063], the `Window`[p965] object wrapped by the browsing context[p1046]'s associated `WindowProxy`[p977] object is changed.

The `WindowProxy`[p977] exotic object must use the ordinary internal methods except where it is explicitly specified otherwise below.

There is no `WindowProxy`[p977] interface object.

Every `WindowProxy`[p977] object has a **[[Window]]** internal slot representing the wrapped `Window`[p965] object.

Note

Although `WindowProxy`[p977] 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`[p977] and `Location`[p980] objects. As long as the `Window`[p965] object remains an ordinary object this is unobservable and can be implemented either way.

7.2.3.1 [[GetPrototypeOf]] ( ) §[p97 7]

1. Let *W* be the value of the [[Window]][p977] internal slot of **this**.
2. If IsPlatformObjectSameOrigin[p963](*W*) is true, then return ! OrdinaryGetPrototypeOf(*W*).
3. Return null.

7.2.3.2 [[SetPrototypeOf]] (*V*) §[p97 7]

1. Return ! SetImmutablePrototype(**this**, *V*).

7.2.3.3 [[IsExtensible]] ( ) §[p97 7]

1. Return true.

7.2.3.4 [[PreventExtensions]] ( ) §[p97 7]

1. Return false.

7.2.3.5 [[GetOwnProperty]] (*P*) §[p97 7]

1. Let *W* be the value of the [[Window]][p977] internal slot of **this**.
2. If *P* is an array index property name:
    1. Let *index* be ! ToUint32(*P*).
    2. Let *children* be the document-tree child navigables[p1041] of *W*'s associated `Document`[p966].

3. Let *value* be undefined.
4. If *index* is less than *children*'s size:
    1. Sort *children* in ascending order, with *navigableA* being less than *navigableB* if *navigableA*'s container[p1038] was inserted into *W*'s associated `Document`[p966] earlier than *navigableB*'s container[p1038] was.
    2. Set *value* to *children*[*index*]'s active `WindowProxy`[p1036].
5. If *value* is undefined:
    1. If IsPlatformObjectSameOrigin[p963](*W*) is true, then return undefined.
    2. Throw a `"SecurityError"` `DOMException`.
6. Return PropertyDescriptor { [[Value]]: *value*, [[Writable]]: false, [[Enumerable]]: true, [[Configurable]]: true }.

3. If IsPlatformObjectSameOrigin[p963](*W*) is true, then return ! OrdinaryGetOwnProperty(*W*, *P*).

> **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 tc39/ecma262 issue #672 for more information. [JAVASCRIPT][p1583]*

4. Let *property* be CrossOriginGetOwnPropertyHelper[p963](*W*, *P*).
5. If *property* is not undefined, then return *property*.
6. If *property* is undefined and *P* is in *W*'s document-tree child navigable target name property set[p972]:
    1. Let *value* be the active `WindowProxy`[p1036] of the named object[p973] of *W* with the name *P*.
    2. Return PropertyDescriptor { [[Value]]: *value*, [[Enumerable]]: false, [[Writable]]: false, [[Configurable]]: true }.

    > **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. Return ? CrossOriginPropertyFallback[p963](*P*).

7.2.3.6 [[DefineOwnProperty]] (*P*, *Desc*) §[p978]

1. Let *W* be the value of the [[Window]][p977] internal slot of **this**.
2. If IsPlatformObjectSameOrigin[p963](*W*) is true:
    1. If *P* is an array index property name, return false.
    2. Return ? OrdinaryDefineOwnProperty(*W*, *P*, *Desc*).

    > **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 tc39/ecma262 issue #672 for more information. [JAVASCRIPT][p1583]*

3. Throw a `"SecurityError"` `DOMException`.

7.2.3.7 [[Get]] (*P*, *Receiver*) §[p978]

1. Let *W* be the value of the [[Window]][p977] internal slot of **this**.
2. Check if an access between two browsing contexts should be reported[p950], given the current global object[p1151]'s browsing context[p966], *W*'s browsing context[p966], *P*, and the current settings object[p1151].

3. If IsPlatformObjectSameOrigin[p963](*W*) is true, then return ? OrdinaryGet(**this**, *P*, *Receiver*).
4. Return ? CrossOriginGet[p964](**this**, *P*, *Receiver*).

Note

this *is passed rather than* W *as OrdinaryGet and CrossOriginGet[p964] will invoke the [[GetOwnProperty]][p977] internal method.*

7.2.3.8 [[Set]] (*P*, *V*, *Receiver*) §[p97 9]

1. Let *W* be the value of the [[Window]][p977] internal slot of **this**.
2. Check if an access between two browsing contexts should be reported[p950], given the current global object[p1151]'s browsing context[p1046], *W*'s browsing context[p1046], *P*, and the current settings object[p1151].
3. If IsPlatformObjectSameOrigin[p963](*W*) is true:
    1. If *P* is an array index property name, then return false.
    2. Return ? OrdinarySet(*W*, *P*, *V*, *Receiver*).
4. Return ? CrossOriginSet[p964](**this**, *P*, *V*, *Receiver*).

Note

this *is passed rather than* W *as CrossOriginSet[p964] will invoke the [[GetOwnProperty]][p977] internal method.*

7.2.3.9 [[Delete]] (*P*) §[p97 9]

1. Let *W* be the value of the [[Window]][p977] internal slot of **this**.
2. If IsPlatformObjectSameOrigin[p963](*W*) is true:
    1. If *P* is an array index property name:
        1. Let *desc* be ! **this**.[[GetOwnProperty]](*P*).
        2. If *desc* is undefined, then return true.
        3. Return false.
    2. Return ? OrdinaryDelete(*W*, *P*).
3. Throw a `"SecurityError"` `DOMException`.

7.2.3.10 [[OwnPropertyKeys]] ( ) §[p97 9]

1. Let *W* be the value of the [[Window]][p977] internal slot of **this**.
2. Let *maxProperties* be *W*'s associated `Document`[p966]'s document-tree child navigables[p1041]'s size.
3. Let *keys* be the range 0 to *maxProperties*, exclusive.
4. If IsPlatformObjectSameOrigin[p963](*W*) is true, then return the concatenation of *keys* and OrdinaryOwnPropertyKeys(*W*).
5. Return the concatenation of *keys* and ! CrossOriginOwnPropertyKeys[p965](*W*).

7.2.4 The Location[p980] interface §[p980]

Each Window[p965] object is associated with a unique instance of a Location[p980] object, allocated when the Window[p965] object is created.

⚠Warning!

The Location[p980] 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[p980] object, run these steps:

1. Let *location* be a new Location[p980] platform object.
2. Let *valueOf* be *location*'s relevant realm[p1151].[[Intrinsics]].[[%Object.prototype.valueOf%]].
3. Perform ! *location*.[[DefineOwnProperty]]("valueOf", { [[Value]]: *valueOf*, [[Writable]]: false, [[Enumerable]]: false, [[Configurable]]: false }).
4. Perform ! *location*.[[DefineOwnProperty]](%Symbol.toPrimitive%[p58], { [[Value]]: undefined, [[Writable]]: false, [[Enumerable]]: false, [[Configurable]]: false }).
5. Set the value of the [[DefaultProperties]][p986] internal slot of *location* to *location*.[[OwnPropertyKeys]]().
6. Return *location*.

Note

The addition of valueOf *and %Symbol.toPrimitive%[p58] own data properties, as well as the fact that all of* Location[p980]*'s IDL attributes are marked* [LegacyUnforgeable]*, is required by legacy code that consulted the* Location[p980] *interface, or stringified it, to determine the document URL, and then used it in a security-sensitive way. In particular, the* valueOf*, %Symbol.toPrimitive%[p58], and* [LegacyUnforgeable] *stringifier mitigations ensure that code such as* foo[location] = bar *or* location + "" *cannot be misdirected.*

For web developers (non-normative)

***document*.location[p980] [= *value*]**

***window*.location[p980] [= *value*]**

Returns a Location[p980] object with the current page's location.

Can be set, to navigate to another page.

The Document[p135] object's **location** getter steps are to return this's relevant global object[p1151]'s Location[p980] object, if this is fully active[p1051], and null otherwise.

The Window[p965] object's **location** getter steps are to return this's Location[p980] object.

Location[p980] objects provide a representation of the URL of their associated Document[p135], as well as methods for navigating[p1063] and reloading[p1077] the associated navigable[p1035].

IDL

```
[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] readonly attribute DOMStringList ancestorOrigins;
};
```

For web developers (non-normative)

***location*.toString()**

***location*.href**[p982]

Returns the Location[p980] object's URL.

Can be set, to navigate to the given URL.

***location*.origin**[p982]

Returns the Location[p980] object's URL's origin.

***location*.protocol**[p982]

Returns the Location[p980] object's URL's scheme.

Can be set, to navigate to the same URL with a changed scheme.

***location*.host**[p982]

Returns the Location[p980] 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**[p983]

Returns the Location[p980] object's URL's host.

Can be set, to navigate to the same URL with a changed host.

***location*.port**[p983]

Returns the Location[p980] object's URL's port.

Can be set, to navigate to the same URL with a changed port.

***location*.pathname**[p984]

Returns the Location[p980] object's URL's path.

Can be set, to navigate to the same URL with a changed path.

***location*.search**[p984]

Returns the Location[p980] 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**[p984]

Returns the Location[p980] 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**[p985]**(*url*)**

Navigates to the given URL.

***location*.replace**[p985]**(*url*)**

Removes the current page from the session history and navigates to the given URL.

***location*.reload**[p985]**()**

Reloads the current page.

***location*.ancestorOrigins**[p986]

Returns a DOMStringList[p121] object listing the origins of the ancestor navigables[p1041]' active documents[p1036].

A Location[p980] object has an associated **relevant Document**, which is its relevant global object[p1151]'s browsing context[p966]'s active document[p1046], if this Location[p980] object's relevant global object[p1151]'s browsing context[p966] is non-null, and null otherwise.

A Location[p980] object has an associated **url**, which is this Location[p980] object's relevant Document[p981]'s URL, if this Location[p980] object's relevant Document[p981] is non-null, and about:blank[p54] otherwise.

To **Location-object navigate** a Location[p980] object *location* to a URL *url*, optionally given a NavigationHistoryBehavior[p995] *historyHandling* (default "auto[p1062]"):

1. Let *navigable* be *location*'s relevant global object[p1151]'s navigable[p966].
2. Let *sourceDocument* be the incumbent global object[p1149]'s associated Document[p966].
3. If *location*'s relevant Document[p981] is not yet completely loaded[p1114], and the incumbent global object[p1149] does not have transient activation[p868], then set *historyHandling* to "replace[p1062]".
4. Navigate[p1063] *navigable* to *url* using *sourceDocument*, with *exceptionsEnabled*[p1063] set to true and *historyHandling*[p1063] set to *historyHandling*.

The **href** getter steps are:

1. If this's relevant Document[p981] is non-null and its origin is not same origin-domain[p940] with the entry settings object[p1148]'s origin[p1144], then throw a "SecurityError" DOMException.
2. Return this's url[p981], serialized.

The href[p982] setter steps are:

1. If this's relevant Document[p981] is null, then return.
2. Let *url* be the result of encoding-parsing a URL[p101] given the given value, relative to the entry settings object[p1148].
3. If *url* is failure, then throw a "SyntaxError" DOMException.
4. Location-object navigate[p981] this to *url*.

Note

The href[p982] setter intentionally has no security check.

The **origin** getter steps are:

1. If this's relevant Document[p981] is non-null and its origin is not same origin-domain[p940] with the entry settings object[p1148]'s origin[p1144], then throw a "SecurityError" DOMException.
2. Return the serialization[p939] of this's url[p981]'s origin.

The **protocol** getter steps are:

1. If this's relevant Document[p981] is non-null and its origin is not same origin-domain[p940] with the entry settings object[p1148]'s origin[p1144], then throw a "SecurityError" DOMException.
2. Return this's url[p981]'s scheme, followed by ":".

The protocol[p982] setter steps are:

1. If this's relevant Document[p981] is null, then return.
2. If this's relevant Document[p981]'s origin is not same origin-domain[p940] with the entry settings object[p1148]'s origin[p1144], then throw a "SecurityError" DOMException.
3. Let *copyURL* be a copy of this's url[p981].
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 navigate[p981] this to *copyURL*.

The **host** getter steps are:

1. If this's relevant Document[p981] is non-null and its origin is not same origin-domain[p940] with the entry settings object[p1148]'s origin[p1144], then throw a "SecurityError" DOMException.
2. Let *url* be this's url[p981].
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.

The host[p982] setter steps are:

1. If this's relevant Document[p981] is null, then return.
2. If this's relevant Document[p981]'s origin is not same origin-domain[p940] with the entry settings object[p1148]'s origin[p1144], then throw a "SecurityError" DOMException.
3. Let *copyURL* be a copy of this's url[p981].
4. If *copyURL* has an opaque path, then return.
5. Basic URL parse the given value, with *copyURL* as *url* and host state as *state override*.
6. Location-object navigate[p981] this to *copyURL*.

The **hostname** getter steps are:

1. If this's relevant Document[p981] is non-null and its origin is not same origin-domain[p940] with the entry settings object[p1148]'s origin[p1144], then throw a "SecurityError" DOMException.
2. If this's url[p981]'s host is null, return the empty string.
3. Return this's url[p981]'s host, serialized.

The hostname[p983] setter steps are:

1. If this's relevant Document[p981] is null, then return.
2. If this's relevant Document[p981]'s origin is not same origin-domain[p940] with the entry settings object[p1148]'s origin[p1144], then throw a "SecurityError" DOMException.
3. Let *copyURL* be a copy of this's url[p981].
4. If *copyURL* has an opaque path, then return.
5. Basic URL parse the given value, with *copyURL* as *url* and hostname state as *state override*.
6. Location-object navigate[p981] this to *copyURL*.

The **port** getter steps are:

1. If this's relevant Document[p981] is non-null and its origin is not same origin-domain[p940] with the entry settings object[p1148]'s origin[p1144], then throw a "SecurityError" DOMException.
2. If this's url[p981]'s port is null, return the empty string.
3. Return this's url[p981]'s port, serialized.

The port[p983] setter steps are:

1. If this's relevant Document[p981] is null, then return.
2. If this's relevant Document[p981]'s origin is not same origin-domain[p940] with the entry settings object[p1148]'s origin[p1144], then throw a "SecurityError" DOMException.
3. Let *copyURL* be a copy of this's url[p981].
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 navigate[p981] this to *copyURL*.

The **`pathname`** getter steps are:

1. If this's relevant `Document`[p981] is non-null and its origin is not same origin-domain[p940] with the entry settings object[p1148]'s origin[p1144], then throw a `"SecurityError"` `DOMException`.
2. Return the result of URL path serializing this `Location`[p980] object's url[p981].

The `pathname`[p984] setter steps are:

1. If this's relevant `Document`[p981] is null, then return.
2. If this's relevant `Document`[p981]'s origin is not same origin-domain[p940] with the entry settings object[p1148]'s origin[p1144], then throw a `"SecurityError"` `DOMException`.
3. Let *copyURL* be a copy of this's url[p981].
4. If *copyURL* has an opaque path, 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 navigate[p981] this to *copyURL*.

The **`search`** getter steps are:

1. If this's relevant `Document`[p981] is non-null and its origin is not same origin-domain[p940] with the entry settings object[p1148]'s origin[p1144], then throw a `"SecurityError"` `DOMException`.
2. If this's url[p981]'s query is either null or the empty string, return the empty string.
3. Return "`?`", followed by this's url[p981]'s query.

The `search`[p984] setter steps are:

1. If this's relevant `Document`[p981] is null, then return.
2. If this's relevant `Document`[p981]'s origin is not same origin-domain[p940] with the entry settings object[p1148]'s origin[p1144], then throw a `"SecurityError"` `DOMException`.
3. Let *copyURL* be a copy of this's url[p981].
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`[p981]'s document's character encoding, *copyURL* as *url*, and query state as *state override*.
6. `Location`-object navigate[p981] this to *copyURL*.

The **`hash`** getter steps are:

1. If this's relevant `Document`[p981] is non-null and its origin is not same origin-domain[p940] with the entry settings object[p1148]'s origin[p1144], then throw a `"SecurityError"` `DOMException`.
2. If this's url[p981]'s fragment is either null or the empty string, return the empty string.
3. Return "#", followed by this's url[p981]'s fragment.

The hash[p984] setter steps are:

1. If this's relevant Document[p981] is null, then return.
2. If this's relevant Document[p981]'s origin is not same origin-domain[p940] with the entry settings object[p1148]'s origin[p1144], then throw a "SecurityError" DOMException.
3. Let *copyURL* be a copy of this's url[p981].
4. Let *thisURLFragment* be *copyURL*'s fragment if it is non-null; otherwise the empty string.
5. Let *input* be the given value with a single leading "#" removed, if any.
6. Set *copyURL*'s fragment to the empty string.
7. Basic URL parse *input*, with *copyURL* as *url* and fragment state as *state override*.
8. If *copyURL*'s fragment is *thisURLFragment*, then return.

   Note

   This bailout is necessary for compatibility with deployed content, which redundantly sets location.hash on scroll. It does not apply to other mechanisms of fragment navigation, such as the location.href[p982] setter or location.assign()[p985].
9. Location-object navigate[p981] this to *copyURL*.

Note

Unlike the equivalent API for the a[p267] and area[p489] elements, the hash[p984] setter does not special case the empty string, to remain compatible with deployed scripts.

The **assign(*url*)** method steps are:

1. If this's relevant Document[p981] is null, then return.
2. If this's relevant Document[p981]'s origin is not same origin-domain[p940] with the entry settings object[p1148]'s origin[p1144], then throw a "SecurityError" DOMException.
3. Let *urlRecord* be the result of encoding-parsing a URL[p101] given *url*, relative to the entry settings object[p1148].
4. If *urlRecord* is failure, then throw a "SyntaxError" DOMException.
5. Location-object navigate[p981] this to *urlRecord*.

The **replace(*url*)** method steps are:

1. If this's relevant Document[p981] is null, then return.
2. Let *urlRecord* be the result of encoding-parsing a URL[p101] given *url*, relative to the entry settings object[p1148].
3. If *urlRecord* is failure, then throw a "SyntaxError" DOMException.
4. Location-object navigate[p981] this to *urlRecord* given "replace[p1062]".

Note

The replace()[p985] method intentionally has no security check.

The **reload()** method steps are:

1. Let *document* be this's relevant Document[p981].
2. If *document* is null, then return.
3. If *document*'s origin is not same origin-domain[p940] with the entry settings object[p1148]'s origin[p1144], then throw a "SecurityError" DOMException.
4. Reload[p1077] *document*'s node navigable[p1036].

A Location[p980] object has an associated **empty DOMStringList** which is a DOMStringList[p121] object whose associated list is an empty list.

Note

This object cannot carry state across navigations because it is only returned when there is no relevant Document[p981], in which case it's not possible to navigate, and it's not possible to return to a non-null relevant Document[p981].

The **ancestorOrigins** getter steps are:

1. If this's relevant Document[p981] is null, then return this's empty DOMStringList[p986].
2. If this's relevant Document[p981]'s origin is not same origin-domain[p940] with the entry settings object[p1148]'s origin[p1144], then throw a "SecurityError" DOMException.
3. Assert: this's relevant Document[p981]'s ancestor origins list[p138] is not null.
4. Otherwise, return this's relevant Document[p981]'s ancestor origins list[p138].

As explained earlier, the Location[p980] exotic object requires additional logic beyond IDL for security purposes. The Location[p980] object must use the ordinary internal methods except where it is explicitly specified otherwise below.

Also, every Location[p980] object has a **[[DefaultProperties]]** internal slot representing its own properties at time of its creation.

7.2.4.1 [[GetPrototypeOf]] ( ) §[p98]

1. If IsPlatformObjectSameOrigin[p963](**this**) is true, then return ! OrdinaryGetPrototypeOf(**this**).
2. Return null.

7.2.4.2 [[SetPrototypeOf]] (*V*) §[p98]

1. Return ! SetImmutablePrototype(**this**, *V*).

7.2.4.3 [[IsExtensible]] ( ) §[p98]

1. Return true.

7.2.4.4 [[PreventExtensions]] ( ) §[p98]

1. Return false.

7.2.4.5 [[GetOwnProperty]] (*P*) §[p98]

1. If IsPlatformObjectSameOrigin[p963](**this**) is true:
    1. Let *desc* be OrdinaryGetOwnProperty(**this**, *P*).
    2. If the value of the [[DefaultProperties]][p986] internal slot of **this** contains *P*, then set *desc*.[[Configurable]] to true.
    3. Return *desc*.
2. Let *property* be CrossOriginGetOwnPropertyHelper[p963](**this**, *P*).
3. If *property* is not undefined, then return *property*.

4. Return ? CrossOriginPropertyFallback[p963](*P*).

7.2.4.6 [[DefineOwnProperty]] (*P*, *Desc*) §[p98]

1. If IsPlatformObjectSameOrigin[p963](**this**) is true:
    1. If the value of the [[DefaultProperties]][p986] internal slot of **this** contains *P*, then return false.
    2. Return ? OrdinaryDefineOwnProperty(**this**, *P*, *Desc*).
2. Throw a "SecurityError" DOMException.

7.2.4.7 [[Get]] (*P*, *Receiver*) §[p98]

1. If IsPlatformObjectSameOrigin[p963](**this**) is true, then return ? OrdinaryGet(**this**, *P*, *Receiver*).
2. Return ? CrossOriginGet[p964](**this**, *P*, *Receiver*).

7.2.4.8 [[Set]] (*P*, *V*, *Receiver*) §[p98]

1. If IsPlatformObjectSameOrigin[p963](**this**) is true, then return ? OrdinarySet(**this**, *P*, *V*, *Receiver*).
2. Return ? CrossOriginSet[p964](**this**, *P*, *V*, *Receiver*).

7.2.4.9 [[Delete]] (*P*) §[p98]

1. If IsPlatformObjectSameOrigin[p963](**this**) is true, then return ? OrdinaryDelete(**this**, *P*).
2. Throw a "SecurityError" DOMException.

7.2.4.10 [[OwnPropertyKeys]] ( ) §[p98]

1. If IsPlatformObjectSameOrigin[p963](**this**) is true, then return OrdinaryOwnPropertyKeys(**this**).
2. Return CrossOriginOwnPropertyKeys[p965](**this**).

7.2.5 The History[p987] interface §[p98]

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 unused, optional USVString? url = null);
  undefined replaceState(any data, DOMString unused, optional USVString? url = null);
};
```

For web developers (non-normative)

history[p988].**length**[p989]

Returns the number of overall session history entries[p1037] for the current traversable navigable[p1037].

history[p988].**scrollRestoration**[p989]

Returns the scroll restoration mode[p1053] of the active session history entry[p1035].

history[p988].**scrollRestoration**[p989] **= *value***

Set the scroll restoration mode[p1053] of the active session history entry[p1035] to *value*.

history[p988].**state**[p989]

Returns the classic history API state[p1053] of the active session history entry[p1035], deserialized into a JavaScript value.

history[p988].**go**[p989]**()**

Reloads the current page.

history[p988].**go**[p989]**(*delta*)**

Goes back or forward the specified number of steps in the overall session history entries[p1037] list for the current traversable navigable[p1037].

A zero delta will reload the current page.

If the delta is out of range, does nothing.

history[p988].**back**[p989]**()**

Goes back one step in the overall session history entries[p1037] list for the current traversable navigable[p1037].

If there is no previous page, does nothing.

history[p988].**forward**[p989]**()**

Goes forward one step in the overall session history entries[p1037] list for the current traversable navigable[p1037].

If there is no next page, does nothing.

history[p988].**pushState**[p989]**(*data*, "")**

Adds a new entry into session history with its classic history API state[p1053] set to a serialization of *data*. The active history entry[p1035]'s URL[p1053] will be copied over and used for the new entry's URL.

(The second parameter exists for historical reasons, and cannot be omitted; passing the empty string is traditional.)

history[p988].**pushState**[p989]**(*data*, "", *url*)**

Adds a new entry into session history with its classic history API state[p1053] set to a serialization of *data*, and with its URL[p1053] set to *url*.

If the current Document[p135] cannot have its URL rewritten[p990] to *url*, a "SecurityError" DOMException will be thrown.

(The second parameter exists for historical reasons, and cannot be omitted; passing the empty string is traditional.)

history[p988].**replaceState**[p989]**(*data*, "")**

Updates the classic history API state[p1053] of the active session history entry[p1035] to a structured clone of *data*.

(The second parameter exists for historical reasons, and cannot be omitted; passing the empty string is traditional.)

history[p988].**replaceState**[p989]**(*data*, "", *url*)**

Updates the classic history API state[p1053] of the active session history entry[p1035] to a structured clone of *data*, and its URL[p1053] to *url*.

If the current Document[p135] cannot have its URL rewritten[p990] to *url*, a "SecurityError" DOMException will be thrown.

(The second parameter exists for historical reasons, and cannot be omitted; passing the empty string is traditional.)

A Document[p135] has a **history object**, a History[p987] object.

The **history** getter steps are to return this's associated Document[p966]'s history object[p988].

Each History[p987] object has **state**, initially null.

Each History[p987] object has a **length**, a non-negative integer, initially 0.

Each History[p987] object has an **index**, a non-negative integer, initially 0.

Note

Although the index[p988] is not directly exposed, it can be inferred from changes to the length[p988] during synchronous navigations. In fact, that is what it's used for.

The **`length`** getter steps are:

1. If this's relevant global object[p1151]'s associated Document[p966] is not fully active[p1051], then throw a "SecurityError" DOMException.
2. Return this's length[p988].

The **`scrollRestoration`** getter steps are:

1. If this's relevant global object[p1151]'s associated Document[p966] is not fully active[p1051], then throw a "SecurityError" DOMException.
2. Return this's relevant global object[p1151]'s navigable[p966]'s active session history entry[p1035]'s scroll restoration mode[p1053].

The scrollRestoration[p989] setter steps are:

1. If this's relevant global object[p1151]'s associated Document[p966] is not fully active[p1051], then throw a "SecurityError" DOMException.
2. Set this's relevant global object[p1151]'s navigable[p966]'s active session history entry[p1035]'s scroll restoration mode[p1053] to the given value.

The **`state`** getter steps are:

1. If this's relevant global object[p1151]'s associated Document[p966] is not fully active[p1051], then throw a "SecurityError" DOMException.
2. Return this's state[p988].

The **`go(delta)`** method steps are to delta traverse[p989] this given *delta*.

The **`back()`** method steps are to delta traverse[p989] this given −1.

The **`forward()`** method steps are to delta traverse[p989] this given +1.

To **delta traverse** a History[p987] object *history* given an integer *delta*:

1. Let *document* be *history*'s relevant global object[p1151]'s associated Document[p966].
2. If *document* is not fully active[p1051], then throw a "SecurityError" DOMException.
3. If *history*'s relevant global object[p1151]'s navigable[p966]'s allowed to perform a navigation or history update[p1036] returns blocked, then return.
4. If *delta* is 0, then reload[p1077] *document*'s node navigable[p1036], and return.
5. Traverse the history by a delta[p1077] given *document*'s node navigable[p1036]'s traversable navigable[p1037], *delta*, and with *sourceDocument*[p1077] set to *document*.

The **`pushState(data, unused, url)`** method steps are to run the shared history push/replace state steps[p989] given this, *data*, *url*, and "push[p1062]".

The **`replaceState(data, unused, url)`** method steps are to run the shared history push/replace state steps[p989] given this, *data*, *url*, and "replace[p1062]".

The **shared history push/replace state steps**, given a History[p987] *history*, a value *data*, a scalar value string-or-null *url*, and a history handling behavior[p1062] *historyHandling*, are:

1. Let *document* be *history*'s relevant global object[p1151]'s associated Document[p966].
2. If *document* is not fully active[p1051], then throw a "SecurityError" DOMException.
3. Let *serializedData* be StructuredSerializeForStorage[p128](*data*). Rethrow any exceptions.

4. Let *newURL* be *document*'s URL.
5. If *url* is not null or the empty string:
    1. Set *newURL* to the result of encoding-parsing a URL given *url*, relative to the relevant settings object of *history*.
    2. If *newURL* is failure, then throw a "SecurityError" DOMException.
    3. If *document* cannot have its URL rewritten to *newURL*, then throw a "SecurityError" DOMException.

    Note

    The special case for the empty string here is historical, and leads to different resulting URLs when comparing code such as `location.href = ""` *(which performs URL parsing on the empty string) versus* `history.pushState(null, "", "")` *(which bypasses it).*
6. If *history*'s relevant global object's navigable's allowed to perform a navigation or history update returns blocked, then return.
7. Let *navigation* be *history*'s relevant global object's navigation API.
8. Let *continue* be the result of firing a push/replace/reload `navigate` event at *navigation* with *navigationType* set to *historyHandling*, *isSameDocument* set to true, *destinationURL* set to *newURL*, and *classicHistoryAPIState* set to *serializedData*.
9. If *continue* is false, then return.
10. Run the URL and history update steps given *document* and *newURL*, with *serializedData* set to *serializedData* and *historyHandling* set to *historyHandling*.

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.)

A `Document` *document* **can have its URL rewritten** to a URL *targetURL* if the following algorithm returns true:

1. Let *documentURL* be *document*'s URL.
2. If *targetURL* and *documentURL* differ in their scheme, username, password, host, or port components, then return false.
3. If *targetURL*'s scheme is an HTTP(S) scheme, then return true.

    Note

    Differences in path, query, and fragment are allowed for `http:` *and* `https:` *URLs.*
4. If *targetURL*'s scheme is "`file`":
    1. If *targetURL* and *documentURL* differ in their path component, then return false.
    2. Return true.

    Note

    Differences in query and fragment are allowed for `file:` *URLs.*
5. If *targetURL* and *documentURL* differ in their path component or query components, then return false.

    Note

    Only differences in fragment are allowed for other types of URLs.
6. Return true.

Example

***document*'s URL**	***targetURL***	**can have its URL rewritten**
`https://example.com/home`	`https://example.com/home#about`	✔

document's URL	*targetURL*	can have its URL rewritten[p990]
https://example.com/home	https://example.com/home?page=shop	✔
https://example.com/home	https://example.com/shop	✔
https://example.com/home	https://user:pass@example.com/home	✘
https://example.com/home	http://example.com/home	✘
file:///path/to/x	file:///path/to/x#hash	✔
file:///path/to/x	file:///path/to/x?search	✔
file:///path/to/x	file:///path/to/y	✘
about:blank	about:blank#hash	✔
about:blank	about:blank?search	✘
about:blank	about:srcdoc	✘
data:text/html,foo	data:text/html,foo#hash	✔
data:text/html,foo	data:text/html,foo?search	✘
data:text/html,foo	data:text/html,bar	✘
data:text/html,foo	data:bar	✘
blob:https://example.com/77becafe-657b-4fdc-8bd3-e83aaa5e8f43	blob:https://example.com/77becafe-657b-4fdc-8bd3-e83aaa5e8f43#hash	✔
blob:https://example.com/77becafe-657b-4fdc-8bd3-e83aaa5e8f43	blob:https://example.com/77becafe-657b-4fdc-8bd3-e83aaa5e8f43?search	✘
blob:https://example.com/77becafe-657b-4fdc-8bd3-e83aaa5e8f43	blob:https://example.com/anything	✘
blob:https://example.com/77becafe-657b-4fdc-8bd3-e83aaa5e8f43	blob:path	✘

Note how only the URL of the Document[p135] matters, and not its origin. They can mismatch in cases like about:blank[p54] Document[p135]s with inherited origins, in sandboxed iframe[p404]s, or when the document.domain[p942] setter has been used.

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:

```
<!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>
```

The problem with such a system is that each time the user clicks, the whole page has to be reloaded. Here instead is another way of doing it, using script:

```
<!DOCTYPE HTML>
<!-- this starts off as https://example.com/line?x=5 -->
<html lang="en">
<title>Line Game - 5</title>
<p>You are at coordinate <span id="coord">5</span> on the line.</p>
<p>
 <a href="?x=6" onclick="go(1); return false;">Advance to 6</a> or
 <a href="?x=4" onclick="go(-1); return false;">retreat to 4</a>?
</p>
<script>
 var currentPage = 5; // prefilled by server
```

```
  function go(d) {
    setupPage(currentPage + d);
    history.pushState(currentPage, "", '?x=' + currentPage);
  }
  onpopstate = function(event) {
    setupPage(event.state);
  }
  function setupPage(page) {
    currentPage = page;
    document.title = 'Line Game - ' + currentPage;
    document.getElementById('coord').textContent = currentPage;
    document.links[0].href = '?x=' + (currentPage+1);
    document.links[0].textContent = 'Advance to ' + (currentPage+1);
    document.links[1].href = '?x=' + (currentPage-1);
    document.links[1].textContent = 'retreat to ' + (currentPage-1);
  }
</script>
```

In systems without script, this still works like the previous example. However, users that *do* have script support can now navigate much faster, since there is no network access for the same experience. Furthermore, contrary to the experience the user would have with just a naïve script-based approach, bookmarking and navigating the session history still work.

In the example above, the *data* argument to the pushState()[p989] 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

Most applications want to use the same scroll restoration mode[p1054] value for all of their history entries. To achieve this they can set the scrollRestoration[p989] attribute as soon as possible (e.g., in the first script[p685] element in the document's head[p180] element) to ensure that any entry added to the history session gets the desired scroll restoration mode.

```
<head>
  <script>
       if ('scrollRestoration' in history)
            history.scrollRestoration = 'manual';
  </script>
</head>
```

7.2.6 The navigation API §[p992]

7.2.6.1 Introduction §[p992]

This section is non-normative.

The navigation API, provided by the global navigation[p995] property, provides a modern and web application-focused way of managing navigations and history entries. It is a successor to the classic location[p980] and history[p988] APIs.

One ability the API provides is inspecting session history entries[p1053]. For example, the following will display the entries' URLs in an ordered list:

```
const ol = document.createElement("ol");
ol.start = 0; // so that the list items' ordinal values match up with the entry indices

for (const entry of navigation.entries()) {
  const li = document.createElement("li");

  if (entry.index < navigation.currentEntry.index) {
```

```
    li.className = "backward";
  } else if (entry.index > navigation.currentEntry.index) {
    li.className = "forward";
  } else {
    li.className = "current";
  }

  li.textContent = entry.url;
  ol.append(li);
}
```

The navigation.entries()[p1001] array contains NavigationHistoryEntry[p999] instances, which have other useful properties in addition to the url[p1000] and index[p1001] properties shown here. Note that the array only contains NavigationHistoryEntry[p999] objects that represent the current navigable[p1035], and thus its contents are not impacted by navigations inside navigable containers[p1038] such as iframe[p404]s, or by navigations of the parent navigable[p1035] in cases where the navigation API is itself being used inside an iframe[p404]. Additionally, it only contains NavigationHistoryEntry[p999] objects representing same-origin[p939] session history entries[p1053], meaning that if the user has visited other origins before or after the current one, there will not be corresponding NavigationHistoryEntry[p999]s.

The navigation API can also be used to navigate, reload, or traverse through the history:

```
<button onclick="navigation.reload()">Reload</button>

<input type="url" id="navigationURL">
<button onclick="navigation.navigate(navigationURL.value)">Navigate</button>

<button id="backButton" onclick="navigation.back()">Back</button>
<button id="forwardButton" onclick="navigation.forward()">Forward</button>

<select id="traversalDestinations"></select>
<button id="goButton" onclick="navigation.traverseTo(traversalDestinations.value)">Traverse To</button>

<script>
backButton.disabled = !navigation.canGoBack;
forwardButton.disabled = !navigation.canGoForward;

for (const entry of navigation.entries()) {
  traversalDestinations.append(new Option(entry.url, entry.key));
}
</script>
```

Note that traversals are again limited to same-origin[p939] destinations, meaning that, for example, navigation.canGoBack[p1002] will be false if the previous session history entry[p1053] is for a page from another origin.

The most powerful part of the navigation API is the navigate[p1575] event, which fires whenever almost any navigation or traversal occurs in the current navigable[p1035]:

```
navigation.onnavigate = event => {
  console.log(event.navigationType); // "push", "replace", "reload", or "traverse"
  console.log(event.destination.url);
  console.log(event.userInitiated);
  // ... and other useful properties
};
```

(The event will not fire for location bar-initiated navigations[p1137], or navigations initiated from other windows, when the destination of the navigation is a new document.)

Much of the time, the event's cancelable property will be true, meaning this event can be canceled using preventDefault():

```
navigation.onnavigate = event => {
```

```
  if (event.cancelable && isDisallowedURL(event.destination.url)) {
    alert(`Please don't go to ${event.destination.url}!`);
    event.preventDefault();
  }
};
```

The cancelable property will be false for some "traverse[p997]" navigations, such as those taking place inside child navigables[p1039], those crossing to new origins, or when the user attempts to traverse again shortly after a previous call to preventDefault() prevented them from doing so.

The NavigateEvent[p1013]'s intercept()[p1016] method allows intercepting a navigation and converting it into a same-document navigation:

```
navigation.addEventListener("navigate", e => {
  // Some navigations, e.g. cross-origin navigations, we cannot intercept.
  // Let the browser handle those normally.
  if (!e.canIntercept) {
    return;
  }

  // Similarly, don't intercept fragment navigations or downloads.
  if (e.hashChange || e.downloadRequest !== null) {
    return;
  }

  const url = new URL(event.destination.url);

  if (url.pathname.startsWith("/articles/")) {
    e.intercept({
      async handler() {
        // The URL has already changed, so show a placeholder while
        // fetching the new content, such as a spinner or loading page.
        renderArticlePagePlaceholder();

        // Fetch the new content and display when ready.
        const articleContent = await getArticleContent(url.pathname, { signal: e.signal });
        renderArticlePage(articleContent);
      }
    });
  }
});
```

Note that the handler[p1014] function can return a promise to represent the asynchronous progress, and success or failure, of the navigation. While the promise is still pending, browser UI can treat the navigation as ongoing (e.g., by presenting a loading spinner). Other parts of the navigation API are also sensitive to these promises, such as the return value of navigation.navigate()[p1004]:

```
const { committed, finished } = await navigation.navigate("/articles/the-navigation-api-is-cool");

// The committed promise will fulfill once the URL has changed, which happens
// immediately (as long as the NavigateEvent wasn't canceled).
await committed;

// The finished promise will fulfill once the Promise returned by handler() has
// fulfilled, which happens once the article is downloaded and rendered. (Or,
// it will reject, if handler() fails along the way).
await finished;
```

7.2.6.2 The Navigation[p995] interface §[p995]

IDL

```
[Exposed=Window]
interface Navigation : EventTarget {
  sequence<NavigationHistoryEntry> entries();
  readonly attribute NavigationHistoryEntry? currentEntry;
  undefined updateCurrentEntry(NavigationUpdateCurrentEntryOptions options);
  readonly attribute NavigationTransition? transition;
  readonly attribute NavigationActivation? activation;

  readonly attribute boolean canGoBack;
  readonly attribute boolean canGoForward;

  NavigationResult navigate(USVString url, optional NavigationNavigateOptions options = {});
  NavigationResult reload(optional NavigationReloadOptions options = {});

  NavigationResult traverseTo(DOMString key, optional NavigationOptions options = {});
  NavigationResult back(optional NavigationOptions options = {});
  NavigationResult forward(optional NavigationOptions options = {});

  attribute EventHandler onnavigate;
  attribute EventHandler onnavigatesuccess;
  attribute EventHandler onnavigateerror;
  attribute EventHandler oncurrententrychange;
};

dictionary NavigationUpdateCurrentEntryOptions {
  required any state;
};

dictionary NavigationOptions {
  any info;
};

dictionary NavigationNavigateOptions : NavigationOptions {
  any state;
  NavigationHistoryBehavior history = "auto";
};

dictionary NavigationReloadOptions : NavigationOptions {
  any state;
};

dictionary NavigationResult {
  Promise<NavigationHistoryEntry> committed;
  Promise<NavigationHistoryEntry> finished;
};

enum NavigationHistoryBehavior {
  "auto",
  "push",
  "replace"
};
```

Each Window[p965] has an associated **navigation API**, which is a Navigation[p995] object. Upon creation of the Window[p965] object, its navigation API[p995] must be set to a new Navigation[p995] object created in the Window[p965] object's relevant realm[p1151].

The **navigation** getter steps are to return this's navigation API[p995].

The following are the event handlers[p1206] (and their corresponding event handler event types[p1209]) that must be supported, as event handler IDL attributes[p1207], by all objects implementing the Navigation[p995] interface:

Event handler[p1206]	Event handler event type[p1209]
onnavigate	navigate[p1575]
onnavigatesuccess	navigatesuccess[p1575]
onnavigateerror	navigateerror[p1575]
oncurrententrychange	currententrychange[p1575]

7.2.6.3 Core infrastructure [§ p996]

Each Navigation[p995] has an associated **entry list**, a list of NavigationHistoryEntry[p999] objects, initially empty.

Each Navigation[p995] has an associated **current entry index**, an integer, initially −1.

The **current entry** of a Navigation[p995] *navigation* is the result of running the following steps:

1. If *navigation* has entries and events disabled[p996], then return null.
2. Assert: *navigation*'s current entry index[p996] is not −1.
3. Return *navigation*'s entry list[p996][*navigation*'s current entry index[p996]].

A Navigation[p995] *navigation* **has entries and events disabled** if the following steps return true:

1. Let *document* be *navigation*'s relevant global object[p1151]'s associated Document[p966].
2. If *document* is not fully active[p1051], then return true.
3. If *document*'s is initial about:blank[p136] is true, then return true.
4. If *document*'s origin is opaque[p939], then return true.
5. Return false.

To **get the navigation API entry index** of a session history entry[p1053] *she* within a Navigation[p995] *navigation*:

1. Let *index* be 0.
2. For each *nhe* of *navigation*'s entry list[p996]:
    1. If *nhe*'s session history entry[p1000] is equal to *she*, then return *index*.
    2. Increment *index* by 1.
3. Return −1.

A key type used throughout the navigation API is the NavigationType[p996] enumeration:

```
IDL
enum NavigationType {
  "push",
  "replace",
  "reload",
  "traverse"
};
```

This captures the main web developer-visible types of "navigations", which (as noted elsewhere[p1060]) do not exactly correspond to this standard's singular navigate[p1063] algorithm. The meaning of each value is the following:

"push"

Corresponds to calls to navigate[p1063] where the history handling behavior[p1062] ends up as "push[p1062]", or to history.pushState()[p989].

"replace"

Corresponds to calls to navigate[p1063] where the history handling behavior[p1062] ends up as "replace[p1062]", or to

`history.replaceState()`[p989].

"reload"

Corresponds to calls to reload[p1077].

"traverse"

Corresponds to calls to traverse the history by a delta[p1077].

Note

The value space of the `NavigationType`[p996] *enumeration is a superset of the value space of the specification-internal history handling behavior*[p1062] *type. Several parts of this standard make use of this overlap, by passing in a history handling behavior*[p1062] *to an algorithm that expects a* `NavigationType`[p996].

7.2.6.4 Initializing and updating the entry list §[p997]

To **initialize the navigation API entries for a new document** given a `Navigation`[p995] *navigation*, a list of session history entries[p1053] *newSHEs*, and a session history entry[p1053] *initialSHE*:

1. Assert: *navigation*'s entry list[p996] is empty.
2. Assert: *navigation*'s current entry index[p996] is −1.
3. If *navigation* has entries and events disabled[p996], then return.
4. For each *newSHE* of *newSHEs*:
    1. Let *newNHE* be a new `NavigationHistoryEntry`[p999] created in the relevant realm[p1151] of *navigation*.
    2. Set *newNHE*'s session history entry[p1000] to *newSHE*.
    3. Append *newNHE* to *navigation*'s entry list[p996].

    Note

    newSHEs *will have originally come from getting session history entries for the navigation API*[p1058]*, and thus each* newSHE *will be contiguous same*[p940] *origin*[p1054] *with* initialSHE.
5. Set *navigation*'s current entry index[p996] to the result of getting the navigation API entry index[p996] of *initialSHE* within *navigation*.

To **update the navigation API entries for reactivation** given a `Navigation`[p995] *navigation*, a list of session history entries[p1053] *newSHEs*, and a session history entry[p1053] *reactivatedSHE*:

1. If *navigation* has entries and events disabled[p996], then return.
2. Let *newNHEs* be a new empty list.
3. Let *oldNHEs* be a clone of *navigation*'s entry list[p996].
4. For each *newSHE* of *newSHEs*:
    1. Let *newNHE* be null.
    2. If *oldNHEs* contains a `NavigationHistoryEntry`[p999] *matchingOldNHE* whose session history entry[p1000] is *newSHE*:
        1. Set *newNHE* to *matchingOldNHE*.
        2. Remove *matchingOldNHE* from *oldNHEs*.
    3. Otherwise:
        1. Set *newNHE* to a new `NavigationHistoryEntry`[p999] created in the relevant realm[p1151] of *navigation*.
        2. Set *newNHE*'s session history entry[p1000] to *newSHE*.
    4. Append *newNHE* to *newNHEs*.

Note

newSHEs *will have originally come from getting session history entries for the navigation API[p1058], and thus each* newSHE *will be contiguous same[p940] origin[p1054] with* reactivatedSHE.

Note

By the end of this loop, all `NavigationHistoryEntry`[p999]*s that remain in* oldNHEs *represent session history entries[p1053] which have been disposed while the* `Document`[p135] *was in bfcache[p1054].*

5. Set *navigation*'s entry list[p661] to *newNHEs*.
6. Set *navigation*'s current entry index[p996] to the result of getting the navigation API entry index[p996] of *reactivatedSHE* within *navigation*.
7. Queue a global task[p1195] on the navigation and traversal task source[p1204] given *navigation*'s relevant global object[p1151] to run the following steps:
    1. For each *disposedNHE* of *oldNHEs*:
        1. Fire an event named `dispose`[p1575] at *disposedNHE*.

Note

We delay these steps by a task to ensure that `dispose`[p1575] *events will fire after the* `pageshow`[p1576] *event. This ensures that* `pageshow`[p1576] *is the first event a page receives upon reactivation[p1101].*

(However, the rest of this algorithm runs before the `pageshow`[p1576] *event fires. This ensures that* `navigation.entries()`[p1001] *and* `navigation.currentEntry`[p1002] *will have correctly-updated values during any* `pageshow`[p1576] *event handlers.)*

To **update the navigation API entries for a same-document navigation** given a `Navigation`[p995] *navigation*, a session history entry[p1053] *destinationSHE*, and a `NavigationType`[p996] *navigationType*:

1. If *navigation* has entries and events disabled[p996], then return.
2. Let *oldCurrentNHE* be the current entry[p996] of *navigation*.
3. Let *disposedNHEs* be a new empty list.
4. If *navigationType* is "`traverse`[p997]":
    1. Set *navigation*'s current entry index[p996] to the result of getting the navigation API entry index[p996] of *destinationSHE* within *navigation*.
    2. Assert: *navigation*'s current entry index[p996] is not −1.

Note

This algorithm is only called for same-document traversals. Cross-document traversals will instead call either initialize the navigation API entries for a new document[p997] or update the navigation API entries for reactivation[p997].

5. Otherwise, if *navigationType* is "`push`[p996]":
    1. Set *navigation*'s current entry index[p996] to *navigation*'s current entry index[p996] + 1.
    2. Let *i* be *navigation*'s current entry index[p996].
    3. While *i* < *navigation*'s entry list[p996]'s size:
        1. Append *navigation*'s entry list[p996][*i*] to *disposedNHEs*.
        2. Set *i* to *i* + 1.
    4. Remove all items in *disposedNHEs* from *navigation*'s entry list[p996].
6. Otherwise, if *navigationType* is "`replace`[p996]":
    1. Append *oldCurrentNHE* to *disposedNHEs*.
7. If *navigationType* is "`push`[p996]" or "`replace`[p996]":

1. Let *newNHE* be a new NavigationHistoryEntry created in the relevant realm of *navigation*.
2. Set *newNHE*'s session history entry to *destinationSHE*.
3. Set *navigation*'s entry list[*navigation*'s current entry index] to *newNHE*.

8. If *navigation*'s ongoing API method tracker is non-null, then notify about the committed-to entry given *navigation*'s ongoing API method tracker and the current entry of *navigation*.

Note

It is important to do this before firing the dispose *or* currententrychange *events, since event handlers could start another navigation, or otherwise change the value of* navigation's *ongoing API method tracker.*

9. Let *navigateEvent* be *navigation*'s ongoing navigate event.
10. Let *apiMethodTracker* be *navigation*'s ongoing API method tracker.
11. Prepare to run script given *navigation*'s relevant settings object.

Note

See the discussion for other navigation API events to understand why we do this.

12. Fire an event named currententrychange at *navigation* using NavigationCurrentEntryChangeEvent, with its navigationType attribute initialized to *navigationType* and its from initialized to *oldCurrentNHE*.
13. For each *disposedNHE* of *disposedNHEs*:
    1. Fire an event named dispose at *disposedNHE*.
14. Run the navigate event intercept commit handler steps given *navigation*, *navigateEvent*, and *apiMethodTracker*.
15. Clean up after running script given *navigation*'s relevant settings object.

In implementations, same-document navigations can cause session history entries to be disposed by falling off the back of the session history entry list. This is not yet handled by the above algorithm (or by any other part of this standard). See issue #8620 to track progress on defining the correct behavior in such cases.

7.2.6.5 The NavigationHistoryEntry interface

IDL

```
[Exposed=Window]
interface NavigationHistoryEntry : EventTarget {
  readonly attribute USVString? url;
  readonly attribute DOMString key;
  readonly attribute DOMString id;
  readonly attribute long long index;
  readonly attribute boolean sameDocument;

  any getState();

  attribute EventHandler ondispose;
};
```

For web developers (non-normative)

***entry*.url**

The URL of this navigation history entry.

This can return null if the entry corresponds to a different Document than the current one (i.e., if sameDocument is false), and that Document was fetched with a referrer policy of "no-referrer" or "origin", since that indicates the Document in question is hiding its URL even from other same-origin pages.

***entry*.key**[p1000]

A user agent-generated random UUID string representing this navigation history entry's place in the navigation history list. This value will be reused by other `NavigationHistoryEntry`[p999] instances that replace this one due to "`replace`[p996]" navigations, and will survive reloads and session restores.

This is useful for navigating back to this entry in the navigation history list, using `navigation.traverseTo(key)`[p1005].

***entry*.id**[p1000]

A user agent-generated random UUID string representing this specific navigation history entry. This value will *not* be reused by other `NavigationHistoryEntry`[p999] instances. This value will survive reloads and session restores.

This is useful for associating data with this navigation history entry using other storage APIs.

***entry*.index**[p1001]

The index of this `NavigationHistoryEntry`[p999] within `navigation.entries()`[p1001], or −1 if the entry is not in the navigation history entry list.

***entry*.sameDocument**[p1001]

Indicates whether or not this navigation history entry is for the same `Document`[p135] as the current one, or not. This will be true, for example, when the entry represents a fragment navigation or single-page app navigation.

***entry*.getState()**[p1001]

Returns the deserialization[p128] of the state stored in this entry, which was added to the entry using `navigation.navigate()`[p1004] or `navigation.updateCurrentEntry()`[p1002]. This state survives reloads and session restores.

Note that in general, unless the state value is a primitive, `entry.getState() !== entry.getState()`, since a fresh deserialization is returned each time.

This state is unrelated to the classic history API's `history.state`[p989].

Each `NavigationHistoryEntry`[p999] has an associated **session history entry**, which is a session history entry[p1053].

The **key** of a `NavigationHistoryEntry`[p999] *nhe* is given by the return value of the following algorithm:

1. If *nhe*'s relevant global object[p1151]'s associated `Document`[p966] is not fully active[p1051], then return the empty string.
2. Return *nhe*'s session history entry[p1000]'s navigation API key[p1053].

The **ID** of a `NavigationHistoryEntry`[p999] *nhe* is given by the return value of the following algorithm:

1. If *nhe*'s relevant global object[p1151]'s associated `Document`[p966] is not fully active[p1051], then return the empty string.
2. Return *nhe*'s session history entry[p1000]'s navigation API ID[p1053].

The **index** of a `NavigationHistoryEntry`[p999] *nhe* is given by the return value of the following algorithm:

1. If *nhe*'s relevant global object[p1151]'s associated `Document`[p966] is not fully active[p1051], then return −1.
2. Return the result of getting the navigation API entry index[p996] of this's session history entry[p1000] within this's relevant global object[p1151]'s navigation API[p995].

The **`url`** getter steps are:

1. Let *document* be this's relevant global object[p1151]'s associated `Document`[p966].
2. If *document* is not fully active[p1051], then return the empty string.
3. Let *she* be this's session history entry[p1000].
4. If *she*'s document[p1053] does not equal *document*, and *she*'s document state[p1053]'s request referrer policy[p1054] is "`no-referrer`" or "`origin`", then return null.
5. Return *she*'s URL[p1053], serialized.

The **`key`** getter steps are to return this's key[p1000].

The **`id`** getter steps are to return this's ID[p1000].

The **index** getter steps are to return this's index[p1000].

The **sameDocument** getter steps are:

1. Let *document* be this's relevant global object[p1151]'s associated Document[p966].
2. If *document* is not fully active[p1051], then return false.
3. Return true if this's session history entry[p1000]'s document[p1053] equals *document*, and false otherwise.

The **getState()** method steps are:

1. If this's relevant global object[p1151]'s associated Document[p966] is not fully active[p1051], then return undefined.
2. Return StructuredDeserialize[p128](this's session history entry[p1000]'s navigation API state[p1053]). Rethrow any exceptions.

> **Note**
>
> *This can in theory throw an exception, if attempting to deserialize a large ArrayBuffer when not enough memory is available.*

The following are the event handlers[p1206] (and their corresponding event handler event types[p1209]) that must be supported, as event handler IDL attributes[p1207], by all objects implementing the NavigationHistoryEntry[p999] interface:

Event handler[p1206]	Event handler event type[p1209]
ondispose	dispose[p1575]

7.2.6.6 The history entry list §[p1001]

For web developers (non-normative)

***entries* = navigation[p995].entries()[p1001]**

Returns an array of NavigationHistoryEntry[p999] instances represent the current navigation history entry list, i.e., all session history entries[p1053] for this navigable[p1035] that are same origin[p940] and contiguous to the current session history entry[p1035].

navigation[p995].currentEntry[p1002]

Returns the NavigationHistoryEntry[p999] corresponding to the current session history entry[p1035].

navigation[p995].updateCurrentEntry[p1002]({ state[p995] })

Updates the navigation API state[p1053] of the current session history entry[p1035], without performing a navigation like navigation.reload()[p1004] would do.

This method is best used to capture updates to the page that have already happened, and need to be reflected into the navigation API state. For cases where the state update is meant to drive a page update, instead use navigation.navigate()[p1004] or navigation.reload()[p1004], which will trigger a navigate[p1575] event.

navigation[p995].canGoBack[p1002]

Returns true if the current current session history entry[p1035] (i.e., currentEntry[p1002]) is not the first one in the navigation history entry list (i.e., in entries()[p1001]). This means that there is a previous session history entry[p1053] for this navigable[p1035], and its document state[p1053]'s origin[p1054] is same origin[p940] with the current Document[p135]'s origin.

navigation[p995].canGoForward[p1002]

Returns true if the current current session history entry[p1035] (i.e., currentEntry[p1002]) is not the last one in the navigation history entry list (i.e., in entries()[p1001]). This means that there is a next session history entry[p1053] for this navigable[p1035], and its document state[p1053]'s origin[p1054] is same origin[p940] with the current Document[p135]'s origin.

The **entries()** method steps are:

1. If this has entries and events disabled[p996], then return the empty list.
2. Return this's entry list[p996].

> **Note**
>
> *Recall that because of Web IDL's sequence type conversion rules, this will create a new JavaScript array object on each*

call. That is, navigation.entries()[p1001] !== navigation.entries()[p1001].

The **currentEntry** getter steps are to return the current entry[p996] of this.

The **updateCurrentEntry(*options*)** method steps are:

1. Let *current* be the current entry[p996] of this.
2. If *current* is null, then throw an "InvalidStateError" DOMException.
3. Let *serializedState* be StructuredSerializeForStorage[p128](*options*["state[p995]"]), rethrowing any exceptions.
4. Set *current*'s session history entry[p1000]'s navigation API state[p1053] to *serializedState*.
5. Fire an event named currententrychange[p1575] at this using NavigationCurrentEntryChangeEvent[p1026], with its navigationType[p1027] attribute initialized to null and its from[p1027] initialized to *current*.

The **canGoBack** getter steps are:

1. If this has entries and events disabled[p996], then return false.
2. Assert: this's current entry index[p996] is not −1.
3. If this's current entry index[p996] is 0, then return false.
4. Return true.

The **canGoForward** getter steps are:

1. If this has entries and events disabled[p996], then return false.
2. Assert: this's current entry index[p996] is not −1.
3. If this's current entry index[p996] is equal to this's entry list[p996]'s size − 1, then return false.
4. Return true.

7.2.6.7 Initiating navigations §[p1002]

For web developers (non-normative)

{ committed[p995], finished[p995] } = navigation[p995].navigate[p1004](*url*)

{ committed[p995], finished[p995] } = navigation[p995].navigate[p1004](*url*, *options*)

Navigates[p1063] the current page to the given *url*. *options* can contain the following values:

- history[p995] can be set to "replace[p1062]" to replace the current session history entry, instead of pushing a new one.
- info[p995] can be set to any value; it will populate the info[p1016] property of the corresponding NavigateEvent[p1013].
- state[p995] can be set to any serializable[p122] value; it will populate the state retrieved by navigation.currentEntry.getState()[p1001] once the navigation completes, for same-document navigations. (It will be ignored for navigations that end up cross-document.)

By default this will perform a full navigation (i.e., a cross-document navigation, unless the given URL differs only in a fragment from the current one). The navigateEvent.intercept()[p1016] method can be used to convert it into a same-document navigation.

The returned promises will behave as follows:

- For navigations that get aborted, both promises will reject with an "AbortError" DOMException.
- For same-document navigations created by using the navigateEvent.intercept()[p1016] method, committed[p995] will fulfill after the navigation commits, and finished[p995] will fulfill or reject according to any promsies returned by handlers passed to intercept()[p1016].

- For other same-document navigations (e.g., non-intercepted fragment navigations[p1071]), both promises will fulfill immediately.
- For cross-document navigations, or navigations that result in 204 or 205 statuses or `Content-Disposition: attachment` header fields from the server (and thus do not actually navigate), both promises will never settle.

In all cases, when the returned promises fulfill, it will be with the NavigationHistoryEntry[p999] that was navigated to.

{ committed[p995], finished[p995] } = navigation[p995].reload[p1004](*options*)

Reloads[p1077] the current page. *options* can contain info[p995] and state[p995], which behave as described above.

The default behavior of performing a from-network-or-cache reload of the current page can be overriden by the using the navigateEvent.intercept()[p1016] method. Doing so will mean this call only updates state or passes along the appropriate info[p995], plus performing whater actions the navigate[p1575] event handlers see fit to carry out.

The returned promises will behave as follows:

- If the reload is intercepted by using the navigateEvent.intercept()[p1016] method, committed[p995] will fulfill after the navigation commits, and finished[p995] will fulfill or reject according to any promsies returned by handlers passed to intercept()[p1016].
- Otherwise, both promises will never settle.

{ committed[p995], finished[p995] } = navigation[p995].traverseTo[p1005](*key*)

{ committed[p995], finished[p995] } = navigation[p995].traverseTo[p1005](*key*, { info[p995] })

Traverses[p1090] to the closest session history entry[p1053] that matches the NavigationHistoryEntry[p999] with the given *key*. info[p995] can be set to any value; it will populate the info[p1016] property of the corresponding NavigateEvent[p1013].

If a traversal to that session history entry[p1053] is already in progress, then this will return the promises for that original traversal, and info[p1016] will be ignored.

The returned promises will behave as follows:

- If there is no NavigationHistoryEntry[p999] in navigation.entries()[p1001] whose key[p1000] matches *key*, both promises will reject with an "InvalidStateError" DOMException.
- For same-document traversals intercepted by the navigateEvent.intercept()[p1016] method, committed[p995] will fulfill as soon as the traversal is processed and navigation.currentEntry[p1002] is updated, and finished[p995] will fulfill or reject according to any promsies returned by the handlers passed to intercept()[p1016].
- For non-intercepted same-document travesals, both promises will fulfill as soon as the traversal is processed and navigation.currentEntry[p1002] is updated.
- For cross-document traversals, including attempted cross-document traversals that end up resulting in a 204 or 205 statuses or `Content-Disposition: attachment` header fields from the server (and thus do not actually traverse), both promises will never settle.

{ committed[p995], finished[p995] } = navigation[p995].back[p1005](*key*)

{ committed[p995], finished[p995] } = navigation[p995].back[p1005](*key*, { info[p995] })

Traverses to the closest previous session history entry[p1053] which results in this navigable[p1035] traversing, i.e., which corresponds to a different NavigationHistoryEntry[p999] and thus will cause navigation.currentEntry[p1002] to change. info[p995] can be set to any value; it will populate the info[p1016] property of the corresponding NavigateEvent[p1013].

If a traversal to that session history entry[p1053] is already in progress, then this will return the promises for that original traversal, and info[p1016] will be ignored.

The returned promises behave equivalently to those returned by traverseTo()[p1005].

{ committed[p995], finished[p995] } = navigation[p995].forward[p1005](*key*)

{ committed[p995], finished[p995] } = navigation[p995].forward[p1005](*key*, { info[p995] })

Traverses to the closest forward session history entry[p1053] which results in this navigable[p1035] traversing, i.e., which corresponds to a different NavigationHistoryEntry[p999] and thus will cause navigation.currentEntry[p1002] to change. info[p995] can be set to any value; it will populate the info[p1016] property of the corresponding NavigateEvent[p1013].

If a traversal to that session history entry[p1053] is already in progress, then this will return the promises for that original traversal, and info[p1016] will be ignored.

The returned promises behave equivalently to those returned by traverseTo()[p1005].

The **navigate(*url*, *options*)** method steps are:

1. Let *urlRecord* be the result of parsing a URL[p101] given *url*, relative to this's relevant settings object[p1151].
2. If *urlRecord* is failure, then return an early error result[p1006] for a "SyntaxError" DOMException.
3. If *urlRecord*'s scheme is "javascript[p1068]", then return an early error result[p1006] for a "NotSupportedError" DOMException.
4. Let *document* be this's relevant global object[p1151]'s associated Document[p966].
5. If *options*["history[p995]"] is "push[p1062]", and the navigation must be a replace[p1062] given *urlRecord* and *document*, then return an early error result[p1006] for a "NotSupportedError" DOMException.
6. Let *state* be *options*["state[p995]"], if it exists; otherwise, undefined.
7. Let *serializedState* be StructuredSerializeForStorage[p128](*state*). If this throws an exception, then return an early error result[p1006] for that exception.

   Note

   It is important to perform this step early, since serialization can invoke web developer code, which in turn might change various things we check in later steps.
8. If *document* is not fully active[p1051], then return an early error result[p1006] for an "InvalidStateError" DOMException.
9. If *document*'s unload counter[p1114] is greater than 0, then return an early error result[p1006] for an "InvalidStateError" DOMException.
10. Let *info* be *options*["info[p995]"], if it exists; otherwise, undefined.
11. Let *apiMethodTracker* be the result of setting up a navigate/reload API method tracker[p1008] for this given *info* and *serializedState*.
12. Navigate[p1063] *document*'s node navigable[p1036] to *urlRecord* using *document*, with *historyHandling*[p1063] set to *options*["history[p995]"], *navigationAPIState*[p1063] set to *serializedState*, and *apiMethodTracker*[p1063] set to *apiMethodTracker*.

    Note

    Unlike location.assign()[p985] *and friends, which are exposed across origin-domain*[p940] *boundaries,* navigation.navigate()[p1004] *can only be accessed by code with direct synchronous access to the* window.navigation[p995] *property. Thus, we avoid the complications about attributing the source document of the navigation, and we don't need to deal with the allowed by sandboxing to navigate*[p1074] *check and its acccompanying* exceptionsEnabled[p1063] *flag. We just treat all navigations as if they come from the* Document[p135] *corresponding to this* Navigation[p995] *object itself (i.e.,* document*).*
13. Return a navigation API method tracker-derived result[p1006] for *apiMethodTracker*.

The **reload(*options*)** method steps are:

1. Let *document* be this's relevant global object[p1151]'s associated Document[p966].
2. Let *serializedState* be StructuredSerializeForStorage[p128](undefined).
3. If *options*["state[p995]"] exists, then set *serializedState* to StructuredSerializeForStorage[p128](*options*["state[p995]"]). If this throws an exception, then return an early error result[p1006] for that exception.

   Note

   It is important to perform this step early, since serialization can invoke web developer code, which in turn might change various things we check in later steps.
4. Otherwise:
    1. Let *current* be the current entry[p996] of this.
    2. If *current* is not null, then set *serializedState* to *current*'s session history entry[p1000]'s navigation API state[p1053].
5. If *document* is not fully active[p1051], then return an early error result[p1006] for an "InvalidStateError" DOMException.
6. If *document*'s unload counter[p1114] is greater than 0, then return an early error result[p1006] for an "InvalidStateError"

DOMException.

7. Let *info* be *options*["info"], if it exists; otherwise, undefined.
8. Let *apiMethodTracker* be the result of setting up a navigate/reload API method tracker for this given *info* and *serializedState*.
9. Reload *document*'s node navigable with *navigationAPIState* set to *serializedState* and *apiMethodTracker* set to *apiMethodTracker*.
10. Return a navigation API method tracker-derived result for *apiMethodTracker*.

The **traverseTo(*key*, *options*)** method steps are:

1. If this's current entry index is −1, then return an early error result for an "InvalidStateError" DOMException.
2. If this's entry list does not contain a NavigationHistoryEntry whose session history entry's navigation API key equals *key*, then return an early error result for an "InvalidStateError" DOMException.
3. Return the result of performing a navigation API traversal given this, *key*, and *options*.

The **back(*options*)** method steps are:

1. If this's current entry index is −1 or 0, then return an early error result for an "InvalidStateError" DOMException.
2. Let *key* be this's entry list[this's current entry index − 1]'s session history entry's navigation API key.
3. Return the result of performing a navigation API traversal given this, *key*, and *options*.

The **forward(*options*)** method steps are:

1. If this's current entry index is −1 or is equal to this's entry list's size − 1, then return an early error result for an "InvalidStateError" DOMException.
2. Let *key* be this's entry list[this's current entry index + 1]'s session history entry's navigation API key.
3. Return the result of performing a navigation API traversal given this, *key*, and *options*.

To **perform a navigation API traversal** given a Navigation *navigation*, a string *key*, and a NavigationOptions *options*:

1. Let *document* be *navigation*'s relevant global object's associated Document.
2. If *document* is not fully active, then return an early error result for an "InvalidStateError" DOMException.
3. If *document*'s unload counter is greater than 0, then return an early error result for an "InvalidStateError" DOMException.
4. Let *current* be the current entry of *navigation*.
5. If *key* equals *current*'s session history entry's navigation API key, then return «["committed" → a promise resolved with *current*, "finished" → a promise resolved with *current*]».
6. If *navigation*'s upcoming traverse API method trackers[*key*] exists, then return a navigation API method tracker-derived result for *navigation*'s upcoming traverse API method trackers[*key*].
7. Let *info* be *options*["info"], if it exists; otherwise, undefined.
8. Let *apiMethodTracker* be the result of adding an upcoming traverse API method tracker for *navigation* given *key* and *info*.
9. Let *navigable* be *document*'s node navigable.
10. Let *traversable* be *navigable*'s traversable navigable.
11. Let *sourceSnapshotParams* be the result of snapshotting source snapshot params given *document*.
12. Append the following session history traversal steps to *traversable*:
    1. Let *navigableSHEs* be the result of getting session history entries given *navigable*.
    2. Let *targetSHE* be the session history entry in *navigableSHEs* whose navigation API key is *key*. If no such

entry exists:

1. Queue a global task[p1195] on the navigation and traversal task source[p1204] given *navigation*'s relevant global object[p1151] to reject the finished promise[p1009] for *apiMethodTracker* with an "InvalidStateError" DOMException.
2. Abort these steps.

Note

This path is taken if navigation's *entry list*[p996] *was outdated compared to* navigableSHEs*, which can occur for brief periods while all the relevant threads and processes are being synchronized in reaction to a history change.*

3. If *targetSHE* is *navigable*'s active session history entry[p1035]:
    1. Queue a global task[p1195] on the navigation and traversal task source[p1204] given *navigation*'s relevant global object[p1151] to reject the finished promise[p1009] for *apiMethodTracker* with an "InvalidStateError" DOMException.
    2. Abort these steps.

Note

This can occur if a previously queued[p1056] *traversal already took us to this session history entry*[p1053].

4. Let *result* be the result of applying the traverse history step[p1090] given by *targetSHE*'s step[p1053] to *traversable*, given *sourceSnapshotParams*, *navigable*, and "none[p1062]".
5. If *result* is "canceled-by-beforeunload", then queue a global task[p1195] on the navigation and traversal task source[p1204] given *navigation*'s relevant global object[p1151] to reject the finished promise[p1009] for *apiMethodTracker* with a new "AbortError" DOMException created in *navigation*'s relevant realm[p1151].
6. If *result* is "initiator-disallowed", then queue a global task[p1195] on the navigation and traversal task source[p1204] given *navigation*'s relevant global object[p1151] to reject the finished promise[p1009] for *apiMethodTracker* with a new "SecurityError" DOMException created in *navigation*'s relevant realm[p1151].

Note

When result *is* "canceled-by-beforeunload" *or* "initiator-disallowed"*, the* navigate[p1575] *event was never fired, aborting the ongoing navigation*[p1010] *would not be correct; it would result in a* navigateerror[p1575] *event without a preceding* navigate[p1575] *event.*

In the "canceled-by-navigate" *case,* navigate[p1575] *is fired, but the inner* navigate *event firing algorithm*[p1020] *will take care of aborting the ongoing navigation*[p1010].

13. Return a navigation API method tracker-derived result[p1006] for *apiMethodTracker*.

An **early error result** for an exception *e* is a NavigationResult[p995] dictionary instance given by «["committed[p995]" → a promise rejected with *e*, "finished[p995]" → a promise rejected with *e*]».

To compute the **navigation API method tracker-derived result** for a navigation API method tracker[p1007]-or-null *apiMethodTracker*:

1. If *apiMethodTracker* is pending[p1007], then return an early error result[p1006] for an "AbortError" DOMException.
2. Return a NavigationResult[p995] dictionary instance given by «["committed[p995]" → *apiMethodTracker*'s committed promise[p1007], "finished[p995]" → *apiMethodTracker*'s finished promise[p1007]]».

7.2.6.8 Ongoing navigation tracking §[p1006]

During any given navigation (in the broad sense of the word[p996]), the Navigation[p995] object needs to keep track of the following:

For all navigations

State	Duration	Explanation
The NavigateEvent[p1013]	For the duration of event firing	So that if the navigation is canceled while the event is firing, we can cancel the event

State	Duration	Explanation
The event's abort controller[p1016]	Until all promises returned from handlers passed to `intercept()`[p1016] have settled	So that if the navigation is canceled, we can signal abort
Whether a new element was focused[p881]	Until all promises returned from handlers passed to `intercept()`[p1016] have settled	So that if one was, focus is not reset[p1025]
The `NavigationHistoryEntry`[p999] being navigated to	From when it is determined, until all promises returned from handlers passed to `intercept()`[p1016] have settled	So that we know what to resolve any `committed`[p995] and `finished`[p995] promises with
Any `finished`[p995] promise that was returned	Until all promises returned from handlers passed to `intercept()`[p1016] have settled	So that we can resolve or reject it appropriately

For non-"`traverse`[p997]" navigations

State	Duration	Explanation
Any `state`[p995]	For the duration of event firing	So that we can update the current entry's navigation API state[p1053] if the event finishes firing without being canceled

For "`traverse`[p997]" navigations

State	Duration	Explanation
Any `info`[p995]	Until the task is queued to fire the `navigate`[p1575] event	So that we can use it to fire the `navigate`[p1575] after the trip through the session history traversal queue[p1037].
Any `committed`[p995] promise that was returned	Until the session history is updated (inside that same task)	So that we can resolve or reject it appropriately
Whether `intercept()`[p1016] was called	Until the session history is updated (inside that same task)	So that we can suppress the normal scroll restoration logic in favor of the behavior given by the `scroll`[p1014] option

We also cannot assume there is only a single navigation requested at any given time, due to web developer code such as:

```
const p1 = navigation.navigate(url1).finished;
const p2 = navigation.navigate(url2).finished;
```

That is, in this scenario, we need to ensure that while navigating to `url2`, we still have the promise `p1` around so that we can reject it. We can't just get rid of any ongoing navigation promises the moment the second call to `navigate()`[p1004] happens.

We end up accomplishing all this by associating the following with each `Navigation`[p995]:

- **Ongoing `navigate` event**, a `NavigateEvent`[p1013] or null, initially null.
- **Focus changed during ongoing navigation**, a boolean, initially false.
- **Suppress normal scroll restoration during ongoing navigation**, a boolean, initially false.
- **Ongoing API method tracker**, a navigation API method tracker[p1007] or null, initially null.
- **Upcoming traverse API method trackers**, an ordered map from strings to navigation API method trackers[p1007], initially empty.

Note

The state here that is not stored in navigation API method trackers[p1007] is state which needs to be tracked even for navigations that are not initiated via navigation API methods.

A **navigation API method tracker** is a struct with the following items:

- A **navigation object**, a `Navigation`[p995]
- A **key**, a string or null
- An **info**, a JavaScript value
- A **serialized state**, a serialized state[p1053] or null
- A **committed-to entry**, a `NavigationHistoryEntry`[p999] or null
- A **committed promise**, a promise
- A **finished promise**, a promise
- A **pending**, a boolean.

All this state is then managed via the following algorithms.

To **set up a navigate/reload API method tracker** given a `Navigation`[p995] *navigation*, a JavaScript value *info*, and a serialized state[p1053]-or-null *serializedState*:

1. Let *committedPromise* and *finishedPromise* be new promises created in *navigation*'s relevant realm[p1151].
2. Mark as handled *finishedPromise*.

¶p1008

Note

The web developer doesn't necessarily care about finishedPromise *being rejected:*

- *They might only care about* committedPromise.
- *They could be doing multiple synchronous navigations within the same task, in which case all but the last will be aborted (causing their* finishedPromise *to reject). This could be an application bug, but also could just be an emergent feature of disparate parts of the application overriding each others' actions.*
- *They might prefer to listen to other transition-failure signals instead of* finishedPromise*, e.g., the* `navigateerror`[p1575] *event, or the* `navigation.transition.finished`[p1012] *promise.*

As such, we mark it as handled to ensure that it never triggers `unhandledrejection`[p1576] *events.*

3. Return a new navigation API method tracker[p1007] with:

navigation object[p1007]
navigation
key[p1007]
null
info[p1007]
info
serialized state[p1007]
serializedState
committed-to entry[p1007]
null
committed promise[p1007]
committedPromise
finished promise[p1007]
finishedPromise
pending[p1007]
false if *navigation* has entries and events disabled[p996]; otherwise true

To **add an upcoming traverse API method tracker** given a `Navigation`[p995] *navigation*, a string *destinationKey*, and a JavaScript value *info*:

1. Let *committedPromise* and *finishedPromise* be new promises created in *navigation*'s relevant realm[p1151].
2. Mark as handled *finishedPromise*.

Note

See the previous discussion[p1008] about why this is done.

3. Let *apiMethodTracker* be a new navigation API method tracker[p1007] with:

navigation object[p1007]
navigation
key[p1007]
destinationKey
info[p1007]
info
serialized state[p1007]
null
committed-to entry[p1007]
null

committed promise[p1007]
committedPromise
finished promise[p1007]
finishedPromise
pending[p1007]
false

4. Set *navigation*'s upcoming traverse API method trackers[p1007][*destinationKey*] to *apiMethodTracker*.
5. Return *apiMethodTracker*.

To **clean up** a navigation API method tracker[p1007] *apiMethodTracker*:

1. Let *navigation* be *apiMethodTracker*'s navigation object[p1007].
2. If *navigation*'s ongoing API method tracker[p1007] is *apiMethodTracker*, then set *navigation*'s ongoing API method tracker[p1007] to null.
3. Otherwise:
    1. Let *key* be *apiMethodTracker*'s key[p1007].
    2. Assert: *key* is not null.
    3. Assert: *navigation*'s upcoming traverse API method trackers[p1007][*key*] exists.
    4. Remove *navigation*'s upcoming traverse API method trackers[p1007][*key*].

To **notify about the committed-to entry** given a navigation API method tracker[p1007] *apiMethodTracker* and a `NavigationHistoryEntry`[p999] *nhe*:

1. Set *apiMethodTracker*'s committed-to entry[p1007] to *nhe*.
2. If *apiMethodTracker*'s serialized state[p1007] is not null, then set *nhe*'s session history entry[p1000]'s navigation API state[p1053] to *apiMethodTracker*'s serialized state[p1007].

    Note

    If it's null, then we're traversing to nhe *via* `navigation.traverseTo()`[p1005]*, which does not allow changing the state.*

    Note

    At this point, apiMethodTracker's *serialized state*[p1007] *is no longer needed. Implementations might want to clear it out to avoid keeping it alive for the lifetime of the navigation API method tracker*[p1007].

3. Resolve *apiMethodTracker*'s committed promise[p1007] with *nhe*.

    Note

    At this point, apiMethodTracker's *committed promise*[p1007] *is only needed in cases where it has not yet been returned to author code. Implementations might want to clear it out to avoid keeping it alive for the lifetime of the navigation API method tracker*[p1007].

To **resolve the finished promise** for a navigation API method tracker[p1007] *apiMethodTracker*:

1. Assert: *apiMethodTracker*'s committed-to entry[p1007] is not null.
2. Resolve *apiMethodTracker*'s finished promise[p1007] with its committed-to entry[p1007].
3. Clean up[p1009] *apiMethodTracker*.

To **reject the finished promise** for a navigation API method tracker[p1007] *apiMethodTracker* with a JavaScript value *exception*:

1. Reject *apiMethodTracker*'s committed promise[p1007] with *exception*.

    Note

    This will do nothing if apiMethodTracker's *committed promise*[p1007] *was previously resolved via notify about the committed-to entry*[p1009].

2. Reject *apiMethodTracker*'s finished promise[p1007] with *exception*.
3. Clean up[p1009] *apiMethodTracker*.

To **abort the ongoing navigation** given a `Navigation`[p995] *navigation* and an optional `DOMException` *error*:

1. Let *event* be *navigation*'s ongoing `navigate` event[p1007].
2. Assert: *event* is not null.
3. Set *navigation*'s focus changed during ongoing navigation[p1007] to false.
4. Set *navigation*'s suppress normal scroll restoration during ongoing navigation[p1007] to false.
5. If *error* was not given, then let *error* be a new `"AbortError"` `DOMException` created in *navigation*'s relevant realm[p1151].
6. If *event*'s dispatch flag is set, then set *event*'s canceled flag to true.
7. Abort[p1010] *event* given *error*.

To **abort a `NavigateEvent`** *event* given *reason*:

1. Let *navigation* be *event*'s relevant global object[p1151]'s navigation API[p995].
2. Set *navigation*'s ongoing `navigate` event[p1007] to null.
3. Signal abort on *event*'s abort controller[p1016] given *reason*.
4. Let *errorInfo* be the result of extracting error information[p1167] from *reason*.

> **Note**
> *For example, if this algorithm is reached because of a call to `window.stop()`[p971], these properties would probably end up initialized based on the line of script that called `window.stop()`[p971]. But if it's because the user clicked the stop button, these properties would probably end up with default values like the empty string or 0.*

5. If *navigation*'s ongoing API method tracker[p1007] is non-null, then reject the finished promise[p1009] for *apiMethodTracker* with *reason*.
6. Fire an event named `navigateerror`[p1575] at *navigation* using `ErrorEvent`[p1168], with additional attributes initialized according to *errorInfo*.
7. If *navigation*'s transition[p1012] is null, then return.
8. Reject *navigation*'s transition[p1012]'s committed promise[p1012] with *reason*.
9. Reject *navigation*'s transition[p1012]'s finished promise[p1012] with *reason*.
10. Set *navigation*'s transition[p1012] to null.

To **inform the navigation API about aborting navigation** in a navigable[p1035] *navigable*:

1. If this algorithm is running on *navigable*'s active window[p1036]'s relevant agent[p1141]'s event loop[p1193], then continue on to the following steps. Otherwise, queue a global task[p1195] on the navigation and traversal task source[p1204] given *navigable*'s active window[p1036] to run the following steps.
2. Let *navigation* be *navigable*'s active window[p1036]'s navigation API[p995].
3. While *navigation*'s ongoing `navigate` event[p1007] is not null:
    1. Abort the ongoing navigation[p1010] given *navigation*.

> **Note**
> *If there is an ongoing cross-document navigation, this means it will be signaled to the navigation API as aborted, e.g., by firing `navigateerror`[p1575] events. This is somewhat accurate, since the next navigation the `Document`[p135] experiences will be this same-document navigation, so a developer which was expecting the next navigation completion to be that of the cross-document navigation gets a useful signal that this did not happen. However, it is also somewhat inaccurate, as the browser will continue to process the ongoing cross-document navigation (applying it after this same-document one*

synchronously finishes).

Ultimately, the navigation API gets a bit messy with overlapping cross- and same-document navigations, as the ongoing navigation tracking[p1006] machinery and APIs are built to expose only a single ongoing navigation. Web developers will be best-served if they do not create such overlapping situations, e.g., by awaiting promises returned from navigation.navigate()[p1004] before starting new navigations.

Note

This is a loop, since abort the ongoing navigation[p1010] can run JavaScript (e.g., via the navigateerror[p1575] event), which might start a new navigation. Since such a newly-started navigation will be superseded by the completion of this navigation, it gets signaled to the navigation API as aborted.

To **inform the navigation API about child navigable destruction** given a navigable[p1035] *navigable*:

1. Inform the navigation API about aborting navigation[p1010] in *navigable*.
2. Let *navigation* be *navigable*'s active window[p1036]'s navigation API[p995].
3. Let *traversalAPIMethodTrackers* be a clone of *navigation*'s upcoming traverse API method trackers[p1007].
4. For each *apiMethodTracker* of *traversalAPIMethodTrackers*: reject the finished promise[p1009] for *apiMethodTracker* with a new "AbortError" DOMException created in *navigation*'s relevant realm[p1151].

The ongoing navigation concept is most-directly exposed to web developers through the navigation.transition[p1012] property, which is an instance of the NavigationTransition[p1011] interface:

IDL

```
[Exposed=Window]
interface NavigationTransition {
  readonly attribute NavigationType navigationType;
  readonly attribute NavigationHistoryEntry from;
  readonly attribute NavigationDestination to;
  readonly attribute Promise<undefined> committed;
  readonly attribute Promise<undefined> finished;
};
```

For web developers (non-normative)

navigation[p995].transition[p1012]

A NavigationTransition[p1011] representing any ongoing navigation that hasn't yet reached the navigatesuccess[p1575] or navigateerror[p1575] stage, if one exists; or null, if there is no such transition ongoing.

Since navigation.currentEntry[p1002] (and other properties like location.href[p982]) are updated immediately upon navigation, this navigation.transition[p1012] property is useful for determining when such navigations are not yet fully settled, according to any handlers passed to navigateEvent.intercept()[p1016].

navigation[p995].transition[p1012].navigationType[p1012]

One of "push[p996]", "replace[p996]", "reload[p997]", or "traverse[p997]", indicating what type of navigation this transition is for.

navigation[p995].transition[p1012].from[p1012]

The NavigationHistoryEntry[p999] from which the transition is coming. This can be useful to compare against navigation.currentEntry[p1002].

navigation[p995].transition[p1012].to[p1012]

The NavigationDestination[p1018] of the transition. This can be useful as a way to reflect a current navigation's destination's url[p1018] without having to listen to the navigate[p1575] event.

navigation[p995].transition[p1012].committed[p1012]

A promise which fulfills once the navigation.currentEntry[p1002] and URL change. This occurs after all of its precommit handlers[p1014] are fulfilled. The promise rejects if one or more of the precommit handlers[p1014] rejects.

navigation[p995].transition[p1012].finished[p1012]

A promise which fulfills at the same time as the navigatesuccess[p1575] fires, or rejects at the same time the navigateerror[p1575]

event fires.

Each Navigation[p995] has a **transition**, which is a NavigationTransition[p1011] or null, initially null.

The **transition** getter steps are to return this's transition[p1012].

Each NavigationTransition[p1011] has an associated **navigation type**, which is a NavigationType[p996].

Each NavigationTransition[p1011] has an associated **from entry**, which is a NavigationHistoryEntry[p999].

Each NavigationTransition[p1011] has an associated **destination**, which is a NavigationDestination[p1018].

Each NavigationTransition[p1011] has an associated **committed promise**, which is a promise.

Each NavigationTransition[p1011] has an associated **finished promise**, which is a promise.

The **navigationType** getter steps are to return this's navigation type[p1012].

The **from** getter steps are to return this's from entry[p1012].

The **to** getter steps are to return this's destination[p1012].

The **committed** getter steps are to return this's committed promise[p1012].

The **finished** getter steps are to return this's finished promise[p1012].

7.2.6.9 The NavigationActivation[p1012] interface §[p1012]

IDL

```
[Exposed=Window]
interface NavigationActivation {
  readonly attribute NavigationHistoryEntry? from;
  readonly attribute NavigationHistoryEntry entry;
  readonly attribute NavigationType navigationType;
};
```

For web developers (non-normative)

navigation[p995]**.activation**[p1013]

A NavigationActivation[p1012] containing information about the most recent cross-document navigation, the navigation that "activated" this Document[p135].

While navigation.currentEntry[p1002] and the Document[p135]'s URL can be updated regularly due to same-document navigations, navigation.activation[p1013] stays constant, and its properties are only updated if the Document[p135] is reactivated[p1101] from history.

navigation[p995]**.activation**[p1013]**.entry**[p1013]

A NavigationHistoryEntry[p999], equivalent to the value of the navigation.currentEntry[p1002] property at the moment the Document[p135] was activated.

navigation[p995]**.activation**[p1013]**.from**[p1013]

A NavigationHistoryEntry[p999], representing the Document[p135] that was active right before the current Document[p135]. This will have a value null in case the previous Document[p135] was not same origin[p940] with this one or if it was the initial about:blank[p136] Document[p135].

There are some cases in which either the from[p1013] or entry[p1013] NavigationHistoryEntry[p999] objects would not be viable targets for the traverseTo()[p1005] method, as they might not be retained in history. For example, the Document[p135] can be activated using location.replace()[p985] or its initial entry could be replaced by history.replaceState()[p989]. However, those entries' url[p1000] property and getState()[p1001] method are still accessible.

navigation[p995]**.activation**[p1013]**.navigationType**[p1013]

One of "push[p996]", "replace[p996]", "reload[p997]", or "traverse[p997]", indicating what type of navigation activated this Document[p135].

Each Navigation[p995] has an associated **activation**, which is null or a NavigationActivation[p1012] object, initially null.

Each NavigationActivation[p1012] has:

- **old entry**, null or a NavigationHistoryEntry[p999].
- **new entry**, null or a NavigationHistoryEntry[p999].
- **navigation type**, a NavigationType[p996].

The **activation** getter steps are to return this's activation[p1013].

The **from** getter steps are to return this's old entry[p1013].

The **entry** getter steps are to return this's new entry[p1013].

The **navigationType** getter steps are to return this's navigation type[p1013].

7.2.6.10 The navigate[p1575] event §p10 13

A major feature of the navigation API is the navigate[p1575] event. This event is fired on any navigation (in the broad sense of the word[p996]), allowing web developers to monitor such outgoing navigations. In many cases, the event is cancelable, which allows preventing the navigation from happening. And in others, the navigation can be intercepted and replaced with a same-document navigation by using the intercept()[p1016] method of the NavigateEvent[p1013] class.

7.2.6.10.1 The NavigateEvent[p1013] interface §p10 13

IDL

```
[Exposed=Window]
interface NavigateEvent : Event {
  constructor(DOMString type, NavigateEventInit eventInitDict);

  readonly attribute NavigationType navigationType;
  readonly attribute NavigationDestination destination;
  readonly attribute boolean canIntercept;
  readonly attribute boolean userInitiated;
  readonly attribute boolean hashChange;
  readonly attribute AbortSignal signal;
  readonly attribute FormData? formData;
  readonly attribute DOMString? downloadRequest;
  readonly attribute any info;
  readonly attribute boolean hasUAVisualTransition;
  readonly attribute Element? sourceElement;

  undefined intercept(optional NavigationInterceptOptions options = {});
  undefined scroll();
};

dictionary NavigateEventInit : EventInit {
  NavigationType navigationType = "push";
  required NavigationDestination destination;
  boolean canIntercept = false;
  boolean userInitiated = false;
  boolean hashChange = false;
  required AbortSignal signal;
  FormData? formData = null;
  DOMString? downloadRequest = null;
  any info;
  boolean hasUAVisualTransition = false;
  Element? sourceElement = null;
```

```
};

dictionary NavigationInterceptOptions {
  NavigationPrecommitHandler precommitHandler;
  NavigationInterceptHandler handler;
  NavigationFocusReset focusReset;
  NavigationScrollBehavior scroll;
};

enum NavigationFocusReset {
  "after-transition",
  "manual"
};

enum NavigationScrollBehavior {
  "after-transition",
  "manual"
};

callback NavigationInterceptHandler = Promise<undefined> ();
```

For web developers (non-normative)

***event*.navigationType** p1016

One of "push p996", "replace p996", "reload p997", or "traverse p997", indicating what type of navigation this is.

***event*.destination** p1016

A NavigationDestination p1018 representing the destination of the navigation.

***event*.canIntercept** p1016

True if intercept() p1016 can be called to intercept this navigation and convert it into a same-document navigation, replacing its usual behavior; false otherwise.

Generally speaking, this will be true whenever the current Document p135 can have its URL rewritten p990 to the destination URL, except for in the case of cross-document "traverse p997" navigations, where it will always be false.

***event*.userInitiated** p1016

True if this navigation was due to a user clicking on an a p267 element, submitting a form p532 element, or using the browser UI p1137 to navigate; false otherwise.

***event*.hashChange** p1016

True for a fragment navigation p1071; false otherwise.

***event*.signal** p1016

An AbortSignal which will become aborted if the navigation gets canceled, e.g., by the user pressing their browser's "Stop" button, or by another navigation interrupting this one.

The expected pattern is for developers to pass this along to any async operations, such as fetch(), which they perform as part of handling this navigation.

***event*.formData** p1016

The FormData representing the submitted form entries for this navigation, if this navigation is a "push p996" or "replace p996" navigation representing a POST form submission p658; null otherwise.

(Notably, this will be null even for "reload p997" or "traverse p997" navigations that are revisiting a session history entry p1053 that was originally created from a form submission.)

***event*.downloadRequest** p1016

Represents whether or not this navigation was requested to be a download, by using an a p267 or area p489 element's download p314 attribute:

- If a download was not requested, then this property is null.
- If a download was requested, returns the filename that was supplied as the download p314 attribute's value. (This could be the empty string.)

Note that a download being requested does not always mean that a download will happen: for example, a download might be blocked by browser security policies, or end up being treated as a "push" navigation for unspecified reasons.

Similarly, a navigation might end up being a download even if it was not requested to be one, due to the destination server responding with a `Content-Disposition: attachment` header.

Finally, note that the navigate event will not fire at all for downloads initiated using browser UI affordances, e.g., those created by right-clicking and choosing to save the target of a link.

***event*.info**

An arbitrary JavaScript value passed via one of the navigation API methods which initiated this navigation, or undefined if the navigation was initiated by the user or by a different API.

***event*.hasUAVisualTransition**

Returns true if the user agent performed a visual transition for this navigation before dispatching this event. If true, the best user experience will be given if the author synchronously updates the DOM to the post-navigation state.

***event*.sourceElement**

Returns the Element responsible for this navigation. This can be an a or area element, a submit button, or a submitted form element.

***event*.intercept({ precommitHandler, handler, focusReset, scroll })**

Intercepts this navigation, preventing its normal handling and instead converting it into a same-document navigation of the same type to the destination URL.

The precommitHandler option can be a function that accepts a NavigationPrecommitController and returns a promise. The precommit handler function will run after the navigate event has finished firing, but before the navigation.currentEntry property has been updated. Returning a rejected promise will abort the navigation and its effect, such as updating the URL and session history. After all the precommit handlers are fulfilled, the navigation can proceed to commit and call the rest of the handlers. The precommitHandler option can only be passed when the event is cancelable: trying to pass a precommitHandler to a non-cancelable NavigateEvent will throw a "SecurityError" DOMException.

The handler option can be a function that returns a promise. The handler function will run after the navigate event has finished firing and the navigation.currentEntry property has been updated. This returned promise is used to signal the duration, and success or failure, of the navigation. After it settles, the browser signals to the user (e.g., via a loading spinner UI, or assistive technology) that the navigation is finished. Additionally, it fires navigatesuccess or navigateerror events as appropriate, which other parts of the web application can respond to.

By default, using this method will cause focus to reset when any handlers' returned promises settle. Focus will be reset to the first element with the autofocus attribute set, or the body element if the attribute isn't present. The focusReset option can be set to "manual" to avoid this behavior.

By default, using this method will delay the browser's scroll restoration logic for "traverse" or "reload" navigations, or its scroll-reset/scroll-to-a-fragment logic for "push" or "replace" navigations, until any handlers' returned promises settle. The scroll option can be set to "manual" to turn off any browser-driven scroll behavior entirely for this navigation, or scroll() can be called before the promise settles to trigger this behavior early.

This method will throw a "SecurityError" DOMException if canIntercept is false, or if isTrusted is false. It will throw an "InvalidStateError" DOMException if not called synchronously, during event dispatch.

***event*.scroll()**

For "traverse" or "reload" navigations, restores the scroll position using the browser's usual scroll restoration logic.

For "push" or "replace" navigations, either resets the scroll position to the top of the document or scrolls to the fragment specified by destination.url if there is one.

If called more than once, or called after automatic post-transition scroll processing has happened due to the scroll option being left as "after-transition", or called before the navigation has committed, this method will throw an "InvalidStateError" DOMException.

Each NavigateEvent has an **interception state**, which is either "none", "intercepted", "committed", "scrolled", or "finished", initially "none".

Each NavigateEvent has a **navigation precommit handler list**, a list of NavigationPrecommitHandler callbacks, initially empty.

Each NavigateEvent has a **navigation handler list**, a list of NavigationInterceptHandler callbacks, initially empty.

Each NavigateEvent[p1013] has a **focus reset behavior**, a NavigationFocusReset[p1014]-or-null, initially null.

Each NavigateEvent[p1013] has a **scroll behavior**, a NavigationScrollBehavior[p1014]-or-null, initially null.

Each NavigateEvent[p1013] has an **abort controller**, an AbortController-or-null, initially null.

Each NavigateEvent[p1013] has a **classic history API state**, a serialized state[p1053] or null. It is only used in some cases where the event's navigationType[p1016] is "push[p996]" or "replace[p996]", and is set appropriately when the event is fired.

The **navigationType**, **destination**, **canIntercept**, **userInitiated**, **hashChange**, **signal**, **formData**, **downloadRequest**, **info**, **hasUAVisualTransition**, and **sourceElement** attributes must return the values they are initialized to.

The **intercept(*options*)** method steps are:

1. Perform shared checks[p1016] given this.
2. If this's canIntercept[p1016] attribute was initialized to false, then throw a "SecurityError" DOMException.
3. If this's dispatch flag is unset, then throw an "InvalidStateError" DOMException.
4. If *options*["precommitHandler[p1014]"] exists:
    1. If this's cancelable attribute is initialized to false, then throw an "InvalidStateError" DOMException.
    2. Append *options*["precommitHandler[p1014]"] to this's navigation precommit handler list[p1015].
5. Assert: this's interception state[p1015] is either "none" or "intercepted".
6. Set this's interception state[p1015] to "intercepted".
7. If *options*["handler[p1014]"] exists, then append it to this's navigation handler list[p1015].
8. If *options*["focusReset[p1014]"] exists:
    1. If this's focus reset behavior[p1016] is not null, and it is not equal to *options*["focusReset[p1014]"], then the user agent may report a warning to the console indicating that the focusReset[p1014] option for a previous call to intercept()[p1016] was overridden by this new value, and the previous value will be ignored.
    2. Set this's focus reset behavior[p1016] to *options*["focusReset[p1014]"].
9. If *options*["scroll[p1014]"] exists:
    1. If this's scroll behavior[p1016] is not null, and it is not equal to *options*["scroll[p1014]"], then the user agent may report a warning to the console indicating that the scroll[p1014] option for a previous call to intercept()[p1016] was overridden by this new value, and the previous value will be ignored.
    2. Set this's scroll behavior[p1016] to *options*["scroll[p1014]"].

The **scroll()** method steps are:

1. Perform shared checks[p1016] given this.
2. If this's interception state[p1015] is not "committed", then throw an "InvalidStateError" DOMException.
3. Process scroll behavior[p1026] given this.

To **perform shared checks** for a NavigateEvent[p1013] *event*:

1. If *event*'s relevant global object[p1151]'s associated Document[p966] is not fully active[p1051], then throw an "InvalidStateError" DOMException.
2. If *event*'s isTrusted attribute was initialized to false, then throw a "SecurityError" DOMException.
3. If *event*'s canceled flag is set, then throw an "InvalidStateError" DOMException.

7.2.6.10.2 The NavigationPrecommitController interface

```
IDL
[Exposed=Window]
interface NavigationPrecommitController {
  undefined redirect(USVString url, optional NavigationNavigateOptions options = {});
  undefined addHandler(NavigationInterceptHandler handler);
};

callback NavigationPrecommitHandler = Promise<undefined> (NavigationPrecommitController controller);
```

For web developers (non-normative)

***precommitController*.redirect(USVString *url*, NavigationNavigateOptions *options*)**

For "push" or "replace" navigations, sets the destination.url to *url*.

If *options* is given, also sets the info and the resulting NavigationHistoryEntry's state to *options*'s info and state, if they are present. The history option can also switch between "push" or "replace" navigations types.

For "reload" or "traverse" navigations, an "InvalidStateError" will be thrown.

If the current Document cannot have its URL rewritten to *url*, a "SecurityError" DOMException will be thrown.

***precommitController*.addHandler(NavigationInterceptHandler *handler*)**

Adds a NavigationInterceptHandler callback that would be called once the navigation is committed, as if this method was passed to the navigateEvent.intercept() method as a handler.

Each NavigationPrecommitController has a NavigateEvent **event**.

The **redirect(*url*, *options*)** method steps are:

1. Assert: this's event's interception state is not "none".
2. Perform shared checks given this's event.
3. If this's event's interception state is not "intercepted", then throw an "InvalidStateError" DOMException.
4. If this's event's navigationType is neither "push" nor "replace", then throw an "InvalidStateError" DOMException.
5. Let *document* be this's relevant global object's associated Document.
6. Let *destinationURL* be the result of parsing *url* given *document*.
7. If *destinationURL* is failure, then throw a "SyntaxError" DOMException.
8. If *document* cannot have its URL rewritten to *destinationURL*, then throw a "SecurityError" DOMException.
9. If *options*["history"] is "push" or "replace", then set this's event's navigationType to *options*["history"].
10. If *options*["state"] exists:
    1. Let *serializedState* be the result of calling StructuredSerializeForStorage(*options*["state"]). This may throw an exception.
    2. Set this's event's destination's state to *serializedState*.
    3. If this's event's target's ongoing API method tracker is non-null, then set this's event's target's ongoing API method tracker's serialized state to *serializedState*.
11. Set this's event's destination's URL to *destinationURL*.
12. If *options*["info"] exists, then set this's event's info to *options*["info"].

The **addHandler(*handler*,)** method steps are:

1. Assert: this's event's interception state is not "none".
2. Perform shared checks given this's event.

3. If this's event[p1017]'s interception state[p1015] is not "intercepted", then throw an "InvalidStateError" DOMException.

4. Append *handler* to this's event[p1017]'s navigation handler list[p1015].

7.2.6.10.3 The NavigationDestination[p1018] interface §[p1018]

IDL

```
[Exposed=Window]
interface NavigationDestination {
  readonly attribute USVString url;
  readonly attribute DOMString key;
  readonly attribute DOMString id;
  readonly attribute long long index;
  readonly attribute boolean sameDocument;

  any getState();
};
```

For web developers (non-normative)

***event*.destination[p1016].url[p1018]**

The URL being navigated to.

***event*.destination[p1016].key[p1019]**

The value of the key[p1000] property of the destination NavigationHistoryEntry[p999], if this is a "traverse[p997]" navigation, or the empty string otherwise.

***event*.destination[p1016].id[p1019]**

The value of the id[p1000] property of the destination NavigationHistoryEntry[p999], if this is a "traverse[p997]" navigation, or the empty string otherwise.

***event*.destination[p1016].index[p1019]**

The value of the index[p1001] property of the destination NavigationHistoryEntry[p999], if this is a "traverse[p997]" navigation, or −1 otherwise.

***event*.destination[p1016].sameDocument[p1019]**

Indicates whether or not this navigation is to the same Document[p135] as the current one, or not. This will be true, for example, in the case of fragment navigations or history.pushState()[p989] navigations.

Note that this property indicates the original nature of the navigation. If a cross-document navigation is converted into a same-document navigation using navigateEvent.intercept()[p1016], that will not change the value of this property.

***event*.destination[p1016].getState[p1019]()**

For "traverse[p997]" navigations, returns the deserialization[p128] of the state stored in the destination session history entry[p1053].

For "push[p996]" or "replace[p996]" navigations, returns the deserialization[p128] of the state passed to navigation.navigate()[p1004], if the navigation was initiated by that method, or undefined it if it wasn't.

For "reload[p997]" navigations, returns the deserialization[p128] of the state passed to navigation.reload()[p1004], if the reload was initiated by that method, or undefined it if it wasn't.

Each NavigationDestination[p1018] has a **URL**, which is a URL.

Each NavigationDestination[p1018] has an **entry**, which is a NavigationHistoryEntry[p999] or null.

Note

It will be non-null if and only if the NavigationDestination[p1018] corresponds to a "traverse[p997]" navigation.

Each NavigationDestination[p1018] has a **state**, which is a serialized state[p1053].

Each NavigationDestination[p1018] has an **is same document**, which is a boolean.

The **url** getter steps are to return this's URL[p1018], serialized.

The **key** getter steps are:

1. If this's entry[p1018] is null, then return the empty string.
2. Return this's entry[p1018]'s key[p1000].

The **id** getter steps are:

1. If this's entry[p1018] is null, then return the empty string.
2. Return this's entry[p1018]'s ID[p1000].

The **index** getter steps are:

1. If this's entry[p1018] is null, then return −1.
2. Return this's entry[p1018]'s index[p1000].

The **sameDocument** getter steps are to return this's is same document[p1018].

The **getState()** method steps are to return StructuredDeserialize[p128](this's state[p1018]).

7.2.6.10.4 Firing the event §[p1019]

Other parts of the standard fire the navigate[p1575] event, through a series of wrapper algorithms given in this section.

To **fire a traverse navigate event** at a Navigation[p995] *navigation* given a session history entry[p1053] ***destinationSHE*** and an optional user navigation involvement[p1062] ***userInvolvement*** (default "none[p1062]"):

1. Let *event* be the result of creating an event given NavigateEvent[p1013], in *navigation*'s relevant realm[p1151].
2. Set *event*'s classic history API state[p1016] to null.
3. Let *destination* be a new NavigationDestination[p1018] created in *navigation*'s relevant realm[p1151].
4. Set *destination*'s URL[p1018] to *destinationSHE*'s URL[p1053].
5. Let *destinationNHE* be the NavigationHistoryEntry[p999] in *navigation*'s entry list[p996] whose session history entry[p1000] is *destinationSHE*, or null if no such NavigationHistoryEntry[p999] exists.
6. If *destinationNHE* is non-null:
    1. Set *destination*'s entry[p1018] to *destinationNHE*.
    2. Set *destination*'s state[p1018] to *destinationSHE*'s navigation API state[p1053].
7. Otherwise,
    1. Set *destination*'s entry[p1018] to null.
    2. Set *destination*'s state[p1018] to StructuredSerializeForStorage[p128](null).
8. Set *destination*'s is same document[p1018] to true if *destinationSHE*'s document[p1053] is equal to *navigation*'s relevant global object[p1151]'s associated Document[p966]; otherwise false.
9. Return the result of performing the inner navigate event firing algorithm[p1020] given *navigation*, "traverse[p997]", *event*, *destination*, *userInvolvement*, null, null, and null.

To **fire a push/replace/reload navigate event** at a Navigation[p995] *navigation* given a NavigationType[p996] ***navigationType***, a URL ***destinationURL***, a boolean ***isSameDocument***, an optional user navigation involvement[p1062] ***userInvolvement*** (default "none[p1062]"), an optional Element-or-null ***sourceElement*** (default null), an optional entry list[p661]-or-null ***formDataEntryList*** (default null), an optional serialized state[p1053] ***navigationAPIState*** (default StructuredSerializeForStorage[p128](null)), an optional serialized state[p1053]-or-null ***classicHistoryAPIState*** (default null), and an optional navigation API method tracker[p1007]-or-null ***apiMethodTracker***:

1. Let *document* be *navigation*'s relevant global object[p1151]'s associated Document[p966].
2. Inform the navigation API about aborting navigation[p1010] in *document*'s node navigable[p1036].

3. If *navigation* has entries and events disabled[p996], and *apiMethodTracker* is not null:

    1. Set *apiMethodTracker*'s pending[p1007] to false.

    2. Set *apiMethodTracker* to null.

> **Note**
>
> *If* navigation *has entries and events disabled[p996], then* `navigate()`[p1004] *and* `reload()`[p1004] *calls return promises that will never fulfill. We never create a* `NavigationHistoryEntry`[p999] *object for such* `Document`[p135]*s, there is no* `NavigationHistoryEntry`[p999] *to apply* serializedState *to, and there is no* `navigate`[p1575] *event to include* info *with. So, we don't need to track this API method call after all. We need to check this after aborting previous navigations, in case the* `Document`[p135] *has became non-fully active[p1051] as a result of a* `navigateerror`[p1575] *event.*

4. If *document* is not fully active[p1051], then return false.

5. Let *event* be the result of creating an event given `NavigateEvent`[p1013], in *navigation*'s relevant realm[p1151].

6. Set *event*'s classic history API state[p1016] to *classicHistoryAPIState*.

7. Let *destination* be a new `NavigationDestination`[p1018] created in *navigation*'s relevant realm[p1151].

8. Set *destination*'s URL[p1018] to *destinationURL*.

9. Set *destination*'s entry[p1018] to null.

10. Set *destination*'s state[p1018] to *navigationAPIState*.

11. Set *destination*'s is same document[p1018] to *isSameDocument*.

12. Return the result of performing the inner `navigate` event firing algorithm[p1020] given *navigation*, *navigationType*, *event*, *destination*, *userInvolvement*, *sourceElement*, *formDataEntryList*, null, and *apiMethodTracker*.

To **fire a download request `navigate` event** at a `Navigation`[p995] *navigation* given a URL ***destinationURL***, a user navigation involvement[p1062] ***userInvolvement***, an `Element`-or-null ***sourceElement***, and a string ***filename***:

1. Let *event* be the result of creating an event given `NavigateEvent`[p1013], in *navigation*'s relevant realm[p1151].

2. Set *event*'s classic history API state[p1016] to null.

3. Let *destination* be a new `NavigationDestination`[p1018] created in *navigation*'s relevant realm[p1151].

4. Set *destination*'s URL[p1018] to *destinationURL*.

5. Set *destination*'s entry[p1018] to null.

6. Set *destination*'s state[p1018] to StructuredSerializeForStorage[p128](null).

7. Set *destination*'s is same document[p1018] to false.

8. Return the result of performing the inner `navigate` event firing algorithm[p1020] given *navigation*, "`push`[p996]", *event*, *destination*, *userInvolvement*, *sourceElement*, null, and *filename*.

The **inner `navigate` event firing algorithm** consists of the following steps, given a `Navigation`[p995] *navigation*, a `NavigationType`[p996] *navigationType*, a `NavigateEvent`[p1013] *event*, a `NavigationDestination`[p1018] *destination*, a user navigation involvement[p1062] *userInvolvement*, an `Element`-or-null *sourceElement*, an entry list[p661]-or-null *formDataEntryList*, a string-or-null *downloadRequestFilename*, and an optional navigation API method tracker[p1007]-or-null *apiMethodTracker* (default null):

1. If *navigation* has entries and events disabled[p996]:

    1. Assert: *navigation*'s ongoing API method tracker[p1007] is null.

    2. Assert: *navigation*'s upcoming traverse API method trackers[p1007] is empty.

    3. Assert: *apiMethodTracker* is null.

    4. Return true.

> **Note**
>
> *These assertions holds because* `traverseTo()`[p1005]*,* `back()`[p1005]*, and* `forward()`[p1005] *will immediately fail when entries*

and events are disabled (since there are no entries to traverse to).

2. Assert: *navigation*'s ongoing API method tracker[p1007] is null.
3. If *destination*'s entry[p1018] is non-null:
    1. Assert: *apiMethodTracker* is null.

       Note

       apiMethodTracker *is passed as an argument only for* `navigate()`[p1004] *and* `reload()`[p1004] *calls.*
    2. Let *destinationKey* be *destination*'s entry[p1018]'s key[p1000].
    3. Assert: *destinationKey* is not the empty string.
    4. If *navigation*'s upcoming traverse API method trackers[p1007][*destinationKey*] exists:
        1. Set *apiMethodTracker* to *navigation*'s upcoming traverse API method trackers[p1007][*destinationKey*].
        2. Remove *navigation*'s upcoming traverse API method trackers[p1007][*destinationKey*].
4. If *apiMethodTracker* is not null:
    1. Set *navigation*'s ongoing API method tracker[p1007] to *apiMethodTracker*.
    2. Set *apiMethodTracker*'s pending[p1007] to false.
5. Let *navigable* be *navigation*'s relevant global object[p1151]'s navigable[p966].
6. Let *document* be *navigation*'s relevant global object[p1151]'s associated `Document`[p966].
7. If *document* can have its URL rewritten[p990] to *destination*'s URL[p1018], and either *destination*'s is same document[p1018] is true or *navigationType* is not "`traverse`[p997]", then initialize *event*'s `canIntercept`[p1016] to true. Otherwise, initialize it to false.
8. Let *traverseCanBeCanceled* be true if all of the following are true:
    - *navigable* is a top-level traversable[p1037];
    - *destination*'s is same document[p1018] is true; and
    - either *userInvolvement* is not "`browser UI`[p1062]", or *navigation*'s relevant global object[p1151] has history-action activation[p868].

    Otherwise, let it be false.
9. If either:
    - *navigationType* is not "`traverse`[p997]"; or
    - *traverseCanBeCanceled* is true,

    then initialize *event*'s `cancelable` to true. Otherwise, initialize it to false.
10. Initialize *event*'s `type` to "`navigate`[p1575]".
11. Initialize *event*'s `navigationType`[p1016] to *navigationType*.
12. Initialize *event*'s `destination`[p1016] to *destination*.
13. Initialize *event*'s `downloadRequest`[p1016] to *downloadRequestFilename*.
14. If *apiMethodTracker* is not null, then initialize *event*'s `info`[p1016] to *apiMethodTracker*'s info[p1007]. Otherwise, initialize it to undefined.

    Note

    At this point apiMethodTracker's *info*[p1007] *is no longer needed and can be nulled out instead of keeping it alive for the lifetime of the navigation API method tracker*[p1007].
15. Initialize *event*'s `hasUAVisualTransition`[p1016] to true if a visual transition, to display a cached rendered state of the

document's latest entry[p1055], was done by the user agent. Otherwise, initialize it to false.

16. Initialize *event*'s `sourceElement`[p1016] to *sourceElement*.
17. Set *event*'s abort controller[p1016] to a new `AbortController` created in *navigation*'s relevant realm[p1151].
18. Initialize *event*'s `signal`[p1016] to *event*'s abort controller[p1016]'s signal.
19. Let *currentURL* be *document*'s URL.
20. If all of the following are true:
    - *event*'s classic history API state[p1016] is null;
    - *destination*'s is same document[p1018] is true;
    - *destination*'s URL[p1018] equals *currentURL* with *exclude fragments* set to true; and
    - *destination*'s URL[p1018]'s fragment is not identical to *currentURL*'s fragment,

    then initialize *event*'s `hashChange`[p1016] to true. Otherwise, initialize it to false.

    Note

    The first condition here means that `hashChange`[p1016] *will be true for fragment navigations*[p1071]*, but false for cases like* `history.pushState(undefined, "", "#fragment")`.
21. If *userInvolvement* is not "`none`[p1062]", then initialize *event*'s `userInitiated`[p1016] to true. Otherwise, initialize it to false.
22. If *formDataEntryList* is not null, then initialize *event*'s `formData`[p1016] to a new `FormData` created in *navigation*'s relevant realm[p1151], associated to *formDataEntryList*. Otherwise, initialize it to null.
23. Assert: *navigation*'s ongoing `navigate` event[p1007] is null.
24. Set *navigation*'s ongoing `navigate` event[p1007] to *event*.
25. Set *navigation*'s focus changed during ongoing navigation[p1007] to false.
26. Set *navigation*'s suppress normal scroll restoration during ongoing navigation[p1007] to false.
27. Let *dispatchResult* be the result of dispatching *event* at *navigation*.
28. If *dispatchResult* is false:
    1. If *navigationType* is "`traverse`[p997]", then consume history-action user activation[p869] given *navigation*'s relevant global object[p1151].
    2. If *event*'s abort controller[p1016]'s signal is not aborted, then abort the ongoing navigation[p1010] given *navigation*.
    3. Return false.
29. If *event*'s interception state[p1015] is not "`none`":
    1. Let *fromNHE* be the current entry[p996] of *navigation*.
    2. Assert: *fromNHE* is not null.
    3. Set *navigation*'s transition[p1012] to a new `NavigationTransition`[p1011] created in *navigation*'s relevant realm[p1151], with

       navigation type[p1012]
       : *navigationType*

       from entry[p1012]
       : *fromNHE*

       destination[p1012]
       : *event*'s `destination`[p1016]

       committed promise[p1012]
       : a new promise created in *navigation*'s relevant realm[p1151]

       finished promise[p1012]
       : a new promise created in *navigation*'s relevant realm[p1151]

4. Mark as handled *navigation*'s transition[p1012]'s finished promise[p1012].

   Note

   See the discussion about other finished promises[p1008] to understand why this is done.

5. Mark as handled *navigation*'s transition[p1012]'s committed promise[p1012].

30. If *event*'s navigation precommit handler list[p1015] is empty then commit[p1024] *event* given *apiMethodTracker*.
31. Otherwise:
    1. Let *precommitController* be a new NavigationPrecommitController[p1017] created in *navigation*'s relevant realm[p1151], whose event[p1017] is *event*.
    2. Let *precommitPromisesList* be an empty list.
    3. For each *handler* of *event*'s navigation precommit handler list[p1015]:
       1. Append the result of invoking *handler* with « *precommitController* » to *precommitPromisesList*.
    4. Wait for all *precommitPromisesList* with the following success steps: commit[p1024] *event* given *apiMethodTracker*, and the following failure step given *reason*: process navigate event handler failure[p1025] given *event* and *reason*.
32. If *event*'s interception state[p1015] is "none", then return true.
33. Return false.

The **navigate event intercept commit handler steps**, given a Navigation[p995] *navigation*, a NavigateEvent[p1013] object *event*, and a navigation API method tracker[p1007] *apiMethodTracker* are as follows:

1. Let *promisesList* be an empty list.
2. For each *handler* of *event*'s navigation handler list[p1015]:
   1. Append the result of invoking *handler* with an empty arguments list to *promisesList*.
3. If *promisesList*'s size is 0, then set *promisesList* to « a promise resolved with undefined ».

   Note

   There is a subtle timing difference between how waiting for all schedules its success and failure steps when given zero promises versus ≥1 promises. For most uses of waiting for all, this does not matter. However, with this API, there are so many events and promise handlers which could fire around the same time that the difference is pretty easily observable: it can cause the event/promise handler sequence to vary. (Some of the events and promises involved include: navigatesuccess[p1575] */* navigateerror[p1575]*,* currententrychange[p1575]*,* dispose[p1575]*,* apiMethodTracker*'s promises, and the* navigation.transition.finished[p1012] *promise.)*

4. Wait for all of *promisesList*, with the following success steps:
   1. If *event*'s relevant global object[p1151] is not fully active[p1051], then abort these steps.
   2. If *event*'s abort controller[p1016]'s signal is aborted, then abort these steps.
   3. Assert: *event* equals *navigation*'s ongoing navigate event[p1007].
   4. Set *navigation*'s ongoing navigate event[p1007] to null.
   5. Finish[p1025] *event* given true.
   6. If *apiMethodTracker* is non-null, then resolve the finished promise[p1009] for *apiMethodTracker*.
   7. Fire an event named navigatesuccess[p1575] at *navigation*.
   8. If *navigation*'s transition[p1012] is not null, then resolve *navigation*'s transition[p1012]'s finished promise[p1012] with undefined.
   9. Set *navigation*'s transition[p1012] to null.

   and the following failure step given *reason*: process navigate event handler failure[p1025] given *event* and *reason*.

To **commit a navigate event** given a NavigateEvent[p1013] object *event* and a navigation API method tracker[p1007] *apiMethodTracker*:

1. Let *navigation* be *event*'s target.
2. Let *navigable* be *event*'s relevant global object[p1151]'s navigable[p966].
3. If *event*'s relevant global object[p1151]'s associated Document[p966] is not fully active[p1051], then return.
4. If *event*'s abort controller[p1016]'s signal is aborted, then return.
5. Let *endResultIsSameDocument* be true if *event*'s interception state[p1015] is not "`none`" or *event*'s `destination`[p1016]'s is same document[p1018] is true.
6. Prepare to run script[p1166] given *navigation*'s relevant settings object[p1151].

¶[p1024] **Note**

This is done to avoid the JavaScript execution context stack becoming empty right after any `currententrychange`[p1575] *event handlers run as a result of the URL and history update steps[p1078] that could soon happen. If the stack were to become empty at that time, then it would immediately perform a microtask checkpoint[p1200], causing various promise fulfillment handlers to run interleaved with the event handlers and before any handlers passed to* `navigateEvent.intercept()`[p1016]*. This is undesirable since it means promise handler ordering vs.* `currententrychange`[p1575] *event handler ordering vs.* `intercept()`[p1016] *handler ordering would be dependent on whether the navigation is happening with an empty JavaScript execution context stack (e.g., because the navigation was user-initiated) or with a nonempty one (e.g., because the navigation was caused by a JavaScript API call).*

By inserting an otherwise-unnecessary JavaScript execution context onto the stack in this step, we essentially suppress the perform a microtask checkpoint[p1200] algorithm until later, thus ensuring that the sequence is always: `currententrychange`[p1575] *event handlers, then* `intercept()`[p1016] *handlers, then promise handlers.*

7. If *event*'s interception state[p1015] is not "`none`":
    1. Set *event*'s interception state[p1015] to "`committed`".
    2. Switch on *event*'s `navigationType`[p1016]:

        ↪ "**`push`**[p996]"

        ↪ "**`replace`**[p996]"

        Run the URL and history update steps[p1078] given *event*'s relevant global object[p1151]'s associated Document[p966] and *event*'s `destination`[p1016]'s URL[p1018], with *serializedData*[p1078] set to *event*'s classic history API state[p1016] and *historyHandling*[p1078] set to *event*'s `navigationType`[p1016].

        ↪ "**`reload`**[p997]"

        Update the navigation API entries for a same-document navigation[p998] given *navigation*, *navigable*'s active session history entry[p1035], and "`reload`[p997]".

        ↪ "**`traverse`**[p997]"

        1. Set *navigation*'s suppress normal scroll restoration during ongoing navigation[p1007] to true.

            Note

            If event's scroll behavior[p1016] was set to "`after-transition`[p1014]*", then scroll restoration will happen as part of finishing[p1025] the relevant* NavigateEvent[p1013]*. Otherwise, there will be no scroll restoration. That is, no navigation which is intercepted by* `intercept()`[p1016] *goes through the normal scroll restoration process; scroll restoration for such navigations is either done manually, by the web developer, or is done after the transition.*

        2. Let *userInvolvement* be "`none`[p1062]".
        3. If *event*'s `userInitiated`[p1016] is true, then set *userInvolvement* to "`activation`[p1062]".

            Note

            At this point after interception, it is not consequential whether the activation was a result of browser UI.

        4. Append the following session history traversal steps[p1056] to *navigable*'s traversable navigable[p1037]:

1. Resume applying the traverse history step[p1090] given *event*'s destination[p1016]'s entry[p1018]'s session history entry[p1000]'s step[p1053], *navigable*'s traversable navigable[p1037], and *userInvolvement*.

8. If *navigation*'s transition[p1012] is not null, then resolve *navigation*'s transition[p1012]'s committed promise[p1012] with undefined.
9. If *endResultIsSameDocument* is false and *apiMethodTracker* is non-null, then clean up[p1009] *apiMethodTracker*.
10. Clean up after running script[p1166] given *navigation*'s relevant settings object[p1151].

> **Note**
> *Per the previous note[p1024], this stops suppressing any potential promise handler microtasks, causing them to run at this point or later.*

To **process navigate event handler failure** given a NavigateEvent[p1013] object *event* and a *reason*:

1. If *event*'s relevant global object[p1151]'s associated Document[p966] is not fully active[p1051], then return.
2. If *event*'s abort controller[p1016]'s signal is aborted, then return.
3. Assert: *event* is *event*'s relevant global object[p1151]'s navigation API[p995]'s ongoing navigate event[p1007].
4. If *event*'s interception state[p1015] is not "intercepted", then finish[p1025] *event* given false.
5. Abort[p1010] *event* given *reason*.

7.2.6.10.5 Scroll and focus behavior §[p1025]

By calling navigateEvent.intercept()[p1016], web developers can suppress the normal scroll and focus behavior for same-document navigations, instead invoking cross-document navigation-like behavior at a later time. The algorithms in this section are called at those appropriate later points.

To **finish** a NavigateEvent[p1013] *event*, given a boolean *didFulfill*:

1. Assert: *event*'s interception state[p1015] is not "finished".
2. If *event*'s interception state[p1015] is "intercepted":
    1. Assert: *didFulfill* is false.
    2. Assert: *event*'s navigation precommit handler list[p1015] is not empty.

    > **Note**
    > *Only precommit handlers can cancel a navigation before it is committed.*

    3. Set *event*'s interception state[p1015] to "finished".
    4. Return.
3. If *event*'s interception state[p1015] is "none", then return.
4. Potentially reset the focus[p1025] given *event*.
5. If *didFulfill* is true, then potentially process scroll behavior[p1026] given *event*.
6. Set *event*'s interception state[p1015] to "finished".

To **potentially reset the focus** given a NavigateEvent[p1013] *event*:

1. Assert: *event*'s interception state[p1015] is "committed" or "scrolled".
2. Let *navigation* be *event*'s relevant global object[p1151]'s navigation API[p995].
3. Let *focusChanged* be *navigation*'s focus changed during ongoing navigation[p1007].
4. Set *navigation*'s focus changed during ongoing navigation[p1007] to false.

5. If *focusChanged* is true, then return.
6. If *event*'s focus reset behavior[p1016] is "manual[p1014]", then return.

   Note

   If it was left as null, then we treat that as "after-transition[p1014]", and continue onward.

7. Let *document* be *event*'s relevant global object[p1151]'s associated Document[p966].
8. Let *focusTarget* be the autofocus delegate[p881] for *document*.
9. If *focusTarget* is null, then set *focusTarget* to *document*'s body element[p144].
10. If *focusTarget* is null, then set *focusTarget* to *document*'s document element.
11. Run the focusing steps[p881] for *focusTarget*, with *document*'s viewport as the *fallback target*.
12. Move the sequential focus navigation starting point[p884] to *focusTarget*.

To **potentially process scroll behavior** given a NavigateEvent[p1013] *event*:

1. Assert: *event*'s interception state[p1015] is "committed" or "scrolled".
2. If *event*'s interception state[p1015] is "scrolled", then return.
3. If *event*'s scroll behavior[p1016] is "manual[p1014]", then return.

   Note

   If it was left as null, then we treat that as "after-transition[p1014]", and continue onward.

4. Process scroll behavior[p1026] given *event*.

To **process scroll behavior** given a NavigateEvent[p1013] *event*:

1. Assert: *event*'s interception state[p1015] is "committed".
2. Set *event*'s interception state[p1015] to "scrolled".
3. If *event*'s navigationType[p1016] was initialized to "traverse[p997]", then restore scroll position data[p1106] given *event*'s relevant global object[p1151]'s navigable[p966]'s active session history entry[p1035].
4. Otherwise:
   1. Let *document* be *event*'s relevant global object[p1151]'s associated Document[p966].
   2. If *document*'s indicated part[p1104] is null, then scroll to the beginning of the document given *document*. [CSSOMVIEW][p1581]
   3. Otherwise, scroll to the fragment[p1104] given *document*.

7.2.7 Event interfaces §[p10 26]

Note

The NavigateEvent[p1013] interface has its own dedicated section[p1013], due to its complexity.

7.2.7.1 The NavigationCurrentEntryChangeEvent[p1026] interface §[p10 26]

IDL

```
[Exposed=Window]
interface NavigationCurrentEntryChangeEvent : Event {
  constructor(DOMString type, NavigationCurrentEntryChangeEventInit eventInitDict);
```

```
  readonly attribute NavigationType? navigationType;
  readonly attribute NavigationHistoryEntry from;
};

dictionary NavigationCurrentEntryChangeEventInit : EventInit {
  NavigationType? navigationType = null;
  required NavigationHistoryEntry from;
};
```

For web developers (non-normative)

***event*.navigationType** p1027

Returns the type of navigation which caused the current entry to change, or null if the change is due to navigation.updateCurrentEntry() p1002.

***event*.from** p1027

Returns the previous value of navigation.currentEntry p1002, before the current entry changed.

If navigationType p1027 is null or "reload p997", then this value will be the same as navigation.currentEntry p1002. In that case, the event signifies that the contents of the entry changed, even if we did not move to a new entry or replace the current one.

The **navigationType** and **from** attributes must return the values they were initialized to.

MDN

7.2.7.2 The PopStateEvent p1027 interface §p10 27

IDL

```
[Exposed=Window]
interface PopStateEvent : Event {
  constructor(DOMString type, optional PopStateEventInit eventInitDict = {});

  readonly attribute any state;
  readonly attribute boolean hasUAVisualTransition;
};

dictionary PopStateEventInit : EventInit {
  any state = null;
  boolean hasUAVisualTransition = false;
};
```

For web developers (non-normative)

***event*.state** p1027

Returns a copy of the information that was provided to pushState() p989 or replaceState() p989.

***event*.hasUAVisualTransition** p1027

Returns true if the user agent performed a visual transition for this navigation before dispatching this event. If true, the best user experience will be given if the author synchronously updates the DOM to the post-navigation state.

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 p135.

The **hasUAVisualTransition** attribute must return the value it was initialized to.

MDN

7.2.7.3 The HashChangeEvent p1027 interface §p10 27

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**[p1028]

Returns the URL of the session history entry[p1053] that was previously current.

***event*.newURL**[p1028]

Returns the URL of the session history entry[p1053] 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[p1053] 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[p1053] that was traversed to.

7.2.7.4 The PageSwapEvent[p1028] interface §[p1028]

IDL

```
[Exposed=Window]
interface PageSwapEvent : Event {
  constructor(DOMString type, optional PageSwapEventInit eventInitDict = {});
  readonly attribute NavigationActivation? activation;
  readonly attribute ViewTransition? viewTransition;
};

dictionary PageSwapEventInit : EventInit {
  NavigationActivation? activation = null;
  ViewTransition? viewTransition = null;
};
```

For web developers (non-normative)

***event*.activation**[p1028]

A NavigationActivation[p1012] object representing the destination and type of the cross-document navigation. This would be null for cross-origin navigations.

***event*.activation**[p1028]**.entry**[p1013]

A NavigationHistoryEntry[p999], representing the Document[p135] that is about to become active.

***event*.activation**[p1028]**.from**[p1013]

A NavigationHistoryEntry[p999], equivalent to the value of the navigation.currentEntry[p1002] property at the moment the event is fired.

***event*.activation**[p1028]**.navigationType**[p1013]

One of "push[p996]", "replace[p996]", "reload[p997]", or "traverse[p997]", indicating what type of navigation that is about to result in a page swap.

***event*.viewTransition**[p1028]

Returns the ViewTransition object that represents an outbound cross-document view transition, if such transition is active when the event is fired. Otherwise, returns null.

The **activation** and **viewTransition** attributes must return the values they were initialized to.

7.2.7.5 The PageRevealEvent interface

IDL

```
[Exposed=Window]
interface PageRevealEvent : Event {
  constructor(DOMString type, optional PageRevealEventInit eventInitDict = {});
  readonly attribute ViewTransition? viewTransition;
};

dictionary PageRevealEventInit : EventInit {
  ViewTransition? viewTransition = null;
};
```

For web developers (non-normative)

***event*.viewTransition**

Returns the ViewTransition object that represents an inbound cross-document view transition, if such transition is active when the event is fired. Otherwise, returns null.

The **viewTransition** attribute must return the value it was initialized to.

7.2.7.6 The PageTransitionEvent interface

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 the page might be reused if the user navigates back to this page (if the Document's *salvageable* state stays true).

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 iframes that are not *salvageable*
- Active WebSocket objects
- Aborting a Document

The **persisted** attribute must return the value it was initialized to. It represents the context information for the event.

To **fire a page transition event** named *eventName* at a Window *window* with a boolean *persisted*, fire an event named *eventName* at *window*, using PageTransitionEvent, with the persisted attribute initialized to *persisted*, the cancelable attribute initialized to true, the bubbles attribute initialized to true, and *legacy target override flag* set.

Note

The values for cancelable and bubbles don't make any sense, since canceling the event does nothing and it's not possible to bubble past the Window object. They are set to true for historical reasons.

7.2.7.7 The BeforeUnloadEvent[p1030] interface §[p10 30]

IDL

```
[Exposed=Window]
interface BeforeUnloadEvent : Event {
  attribute DOMString returnValue;
};
```

Note

There are no BeforeUnloadEvent[p1030]-specific initialization methods.

The BeforeUnloadEvent[p1030] interface is a legacy interface which allows checking if unloading is canceled[p1074] to be controlled not only by canceling the event, but by setting the returnValue[p1030] attribute to a value besides the empty string. Authors should use the preventDefault() method, or other means of canceling events, instead of using returnValue[p1030].

The **returnValue** attribute controls the process of checking if unloading is canceled[p1074]. 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.

7.2.8 The NotRestoredReasons[p1030] interface §[p10 30]

IDL

```
[Exposed=Window]
interface NotRestoredReasonDetails {
  readonly attribute DOMString reason;
  [Default] object toJSON();
};

[Exposed=Window]
interface NotRestoredReasons {
  readonly attribute USVString? src;
  readonly attribute DOMString? id;
  readonly attribute DOMString? name;
  readonly attribute USVString? url;
  readonly attribute FrozenArray<NotRestoredReasonDetails>? reasons;
  readonly attribute FrozenArray<NotRestoredReasons>? children;
  [Default] object toJSON();
};
```

For web developers (non-normative)

***notRestoredReasonDetails*.reason**[p1031]

Returns a string that explains the reason that prevented the document from being served from back/forward cache[p1054]. See the definition of bfcache blocking details[p137] for the possible string values.

***notRestoredReasons*.src**[p1034]

Returns the src[p405] attribute of the document's node navigable[p1036]'s container[p1038] if it is an iframe[p404] element. This can be null if not set or if it is not an iframe[p404] element.

***notRestoredReasons*.id**[p1034]

Returns the id[p162] attribute of the document's node navigable[p1036]'s container[p1038] if it is an iframe[p404] element. This can be null if not set or if it is not an iframe[p404] element.

***notRestoredReasons*.name**[p1034]

Returns the name[p409] attribute of the document's node navigable[p1036]'s container[p1038] if it is an iframe[p404] element. This can be null if not set or if it is not an iframe[p404] element.

***notRestoredReasons*.url**[p1034]

Returns the document's URL, or null if the document is in a cross-origin `iframe`[p404]. This is reported in addition to `src`[p1034] because it is possible `iframe`[p404] navigated since the original `src`[p405] was set.

***notRestoredReasons*.reasons**[p1034]

Returns an array of `NotRestoredReasonDetails`[p1030] for the document. This is null if the document is in a cross-origin `iframe`[p404].

***notRestoredReasons*.children**[p1034]

Returns an array of `NotRestoredReasons`[p1030] that are for the document's children. This is null if the document is in a cross-origin `iframe`[p404].

A `NotRestoredReasonDetails`[p1030] object has a **backing struct**, a not restored reason details[p1031] or null, initially null.

The **`reason`** getter steps are to return this's backing struct[p1031]'s reason[p1031].

To **create a `NotRestoredReasonDetails` object** given a not restored reason details[p1031] *backingStruct* and a realm *realm*:

1. Let *notRestoredReasonDetails* be a new `NotRestoredReasonDetails`[p1030] object created in *realm*.
2. Set *notRestoredReasonDetails*'s backing struct[p1031] to *backingStruct*.
3. Return *notRestoredReasonDetails*.

A **not restored reason details** is a struct with the following items:

- **reason**, a string, initially empty.

The reason[p1031] is a string that represents the reason that prevented the page from being restored from the back/forward cache[p1054]. The string is one of the following:

"`fetch`"

While unloading[p1114], a fetch initiated by this `Document`[p135] was still ongoing and was canceled, so the page was not in a state that could be stored in the back/forward cache[p1054].

"`navigation-failure`"

The original navigation that created this `Document`[p135] errored, so storing the resulting error document in the back/forward cache[p1054] was prevented.

"`parser-aborted`"

The `Document`[p135] never finished its initial HTML parsing, so storing the unfinished document in the back/forward cache[p1054] was prevented.

"`websocket`"

While unloading[p1114], an open `WebSocket` connection was shut down, so the page was not in a state that could be stored in the back/forward cache[p1054]. [WEBSOCKETS][p1588]

"`lock`"

While unloading[p1114], held locks and lock requests were terminated, so the page was not in a state that could be stored in the back/forward cache[p1054]. [WEBLOCKS][p1588]

"`masked`"

This `Document`[p135] has children that are in a cross-origin `iframe`[p404], and they prevented back/forward cache[p1054]; or this `Document`[p135] could not be back/forward cached[p1054] for user agent-specific reasons, and the user agent has chosen not to use one of the more specific reasons from the list of user-agent specific blocking reasons[p1031].

In addition to the list above, a user agent might choose to expose a reason that prevented the page from being restored from back/forward cache[p1054] for **user-agent specific blocking reasons**. These are one of the following strings:

"`audio-capture`"

The `Document`[p135] requested audio capture permission by using *Media Capture and Streams*'s `getUserMedia()` with audio. [MEDIASTREAM][p1584]

"background-work"

The Document[p135] requested background work by calling SyncManager's register() method, PeriodicSyncManager's register() method, or BackgroundFetchManager's fetch() method.

"broadcastchannel-message"

While the page was stored in back/forward cache[p1054], a BroadcastChannel[p1303] connection on the page received a message and message[p1575] event was fired.

"idbversionchangeevent"

The Document[p135] had a pending IDBVersionChangeEvent while unloading[p1114]. [INDEXEDDB][p1583]

"idledetector"

The Document[p135] had an active IdleDetector while unloading[p1114].

"keyboardlock"

While unloading[p1114], keyboard lock was still active because Keyboard's lock() method was called.

"mediastream"

A MediaStreamTrack was in the live state upon unloading[p1114]. [MEDIASTREAM][p1584]

"midi"

The Document[p135] requested a MIDI permission by calling navigator.requestMIDIAccess().

"modals"

User prompts[p1258] were shown while unloading[p1114].

"navigating"

While unloading[p1114], loading was still ongoing, and so the Document[p135] was not in a state that could be stored in back/forward cache[p1054].

"navigation-canceled"

The navigation request was canceled by calling window.stop()[p971] and the page was not in a state to be stored in the back/forward cache[p1054].

"non-trivial-browsing-context-group"

The browsing context group[p1050] of this Document[p135] had more than one top-level browsing context[p1049].

"otpcredential"

The Document[p135] created an OTPCredential.

"outstanding-network-request"

While unloading[p1114], the Document[p135] had outstanding network requests and was not in a state that could be stored in back/forward cache[p1054].

"paymentrequest"

The Document[p135] had an active PaymentRequest while unloading[p1114]. [PAYMENTREQUEST][p1585]

"pictureinpicturewindow"

The Document[p135] had an active PictureInPictureWindow while unloading[p1114]. [PICTUREINPICTURE][p1585]

"plugins"

The Document[p135] contained plugins[p48].

"request-method-not-get"

The Document[p135] was created from an HTTP request whose method was not `GET`. [FETCH][p1582]

"response-auth-required"

The Document[p135] was created from an HTTP response that required HTTP authentication.

"response-cache-control-no-store"

The Document[p135] was created from an HTTP response whose `Cache-Control` header included the "no-store" token. [HTTP][p1583]

"response-cache-control-no-cache"

The Document[p135] was created from an HTTP response whose `Cache-Control` header included the "no-cache" token. [HTTP][p1583]

"response-keep-alive"

The Document[p135] was created from an HTTP response that contained a `Keep-Alive` header.

"**response-scheme-not-http-or-https**"

The Document[p135] was created from a response whose URL's scheme was not an HTTP(S) scheme. [FETCH][p1582]

"**response-status-not-ok**"

The Document[p135] was created from an HTTP response whose status was not an ok status. [FETCH][p1582]

"**rtc**"

While unloading[p1114], a RTCPeerConnection or RTCDataChannel was shut down, so the page was not in a state that could be stored in the back/forward cache[p1054]. [WEBRTC][p1588]

"**rtc-used-with-cache-control-no-store**"

The Document[p135] was created from an HTTP response whose `Cache-Control` header included the "no-store" token, and it has created a RTCPeerConnection or RTCDataChannel which might be used to receive sensitive information, so the page was not in a state that could be stored in the back/forward cache[p1054]. [HTTP][p1583] [WEBRTC][p1588]

"**sensors**"

The Document[p135] requested sensor access.

"**serviceworker-added**"

The Document[p135]'s service worker client started to be controlled by a ServiceWorker while the page was in back/forward cache[p1054]. [SW][p1587]

"**serviceworker-claimed**"

The Document[p135]'s service worker client's active service worker[p1144] was claimed while the page was in back/forward cache[p1054]. [SW][p1587]

"**serviceworker-postmessage**"

The Document[p135]'s service worker client's active service worker[p1144] received a message while the page was in back/forward cache[p1054]. [SW][p1587]

"**serviceworker-version-activated**"

The Document[p135]'s service worker client's active service worker[p1144]'s version was activated while the page was in back/forward cache[p1054]. [SW][p1587]

"**serviceworker-unregistered**"

The Document[p135]'s service worker client's active service worker[p1144]'s service worker registration was unregistered while the page was in back/forward cache[p1054]. [SW][p1587]

"**sharedworker**"

This Document[p135] was in the owner set[p1322] of a SharedWorkerGlobalScope[p1324].

Note

User agents that support the back/forward cache[p1054] for documents using shared workers can use more specific reasons such as "sharedworker-message"[p1033] or "sharedworker-with-no-active-client"[p1033] instead of this general reason, since the usage of shared workers itself is supported outside of the cases described in those reasons.

"**sharedworker-message**"

While the page was stored in back/forward cache[p1054], a message was received from a SharedWorkerGlobalScope[p1324] whose owner set[p1322] contains this Document[p135].

"**sharedworker-with-no-active-client**"

This Document[p135] was in the owner set[p1322] of a SharedWorkerGlobalScope[p1324] that is not actively needed[p1326].

"**smartcardconnection**"

The Document[p135] had an active SmartCardConnection while unloading[p1114].

"**speechrecognition**"

The Document[p135] had an active SpeechRecognition while unloading[p1114].

"**storageaccess**"

The Document[p135] requested storage access permission by using the Storage Access API.

"**unload-listener**"

The Document[p135] registered an event listener for the unload[p1576] event.

"**video-capture**"

The Document[p135] requested video capture permission by using *Media Capture and Streams*'s getUserMedia() with video. [MEDIASTREAM][p1584]

"**webhid**"

The Document[p135] called the WebHID API's requestDevice() method.

"**webshare**"

The Document[p135] used the Web Share API's navigator.share() method.

"**websocket-used-with-cache-control-no-store**"

The Document[p135] was created from an HTTP response whose `Cache-Control` header included the "no-store" token, and it has created a WebSocket connection which might be used to receive sensitive information, so the page was not in a state that could be stored in the back/forward cache[p1054]. [HTTP][p1583] [WEBSOCKETS][p1588]

"**webtransport**"

While unloading[p1114], an open WebTransport connection was shut down, so the page was not in a state that could be stored in the back/forward cache[p1054]. [WEBTRANSPORT][p1588]

"**webtransport-used-with-cache-control-no-store**"

The Document[p135] was created from an HTTP response whose `Cache-Control` header included the "no-store" token, and it has created a WebTransport connection which might be used to receive sensitive information, so the page was not in a state that could be stored in the back/forward cache[p1054]. [HTTP][p1583] [WEBTRANSPORT][p1588]

"**webxrdevice**"

The Document[p135] created a XRSystem.

A NotRestoredReasons[p1030] object has a **backing struct**, a not restored reasons[p1035] or null, initially null.

A NotRestoredReasons[p1030] object has a **reasons array**, a FrozenArray<NotRestoredReasonDetails[p1030]> or null, initially null.

A NotRestoredReasons[p1030] object has a **children array**, a FrozenArray<NotRestoredReasons[p1030]> or null, initially null.

The **src** getter steps are to return this's backing struct[p1034]'s src[p1035].

The **id** getter steps are to return this's backing struct[p1034]'s id[p1035].

The **name** getter steps are to return this's backing struct[p1034]'s name[p1035].

The **url** getter steps are:

1. If this's backing struct[p1034]'s URL[p1035] is null, then return null.
2. Return this's backing struct[p1034]'s URL[p1035], serialized.

The **reasons** getter steps are to return this's reasons array[p1034].

The **children** getter steps are to return this's children array[p1034].

To **create a NotRestoredReasons object** given a not restored reasons[p1035] *backingStruct* and a realm *realm*:

1. Let *notRestoredReasons* be a new NotRestoredReasons[p1030] object created in *realm*.
2. Set *notRestoredReasons*'s backing struct[p1034] to *backingStruct*.
3. If *backingStruct*'s reasons[p1035] is null, set *notRestoredReasons*'s reasons array[p1034] to null.
4. Otherwise:
    1. Let *reasonsArray* be an empty list.
    2. For each *reason* of *backingStruct*'s reasons[p1035]:
        1. Create a NotRestoredReasonDetails object[p1031] given *reason* and *realm*, and append it to *reasonsArray*.
    3. Set *notRestoredReasons*'s reasons array[p1034] to the result of creating a frozen array given *reasonsArray*.

5. If *backingStruct*'s children[p1035] is null, set *notRestoredReasons*'s children array[p1034] to null.
6. Otherwise:
    1. Let *childrenArray* be an empty list.
    2. For each *child* of *backingStruct*'s children[p1035]:
        1. Create a `NotRestoredReasons` object[p1034] given *child* and *realm* and append it to *childrenArray*.
    3. Set *notRestoredReasons*'s children array[p1034] to the result of creating a frozen array given *childrenArray*.
7. Return *notRestoredReasons*.

A **not restored reasons** is a struct with the following items:

- **src**, a scalar value string or null, initially null.
- **id**, a string or null, initially null.
- **name**, a string or null, initially null.
- **url**, a URL or null, initially null.
- **reasons**, null or a list of not restored reason details[p1031], initially null.
- **children**, null or a list of not restored reasons[p1035], initially null.

A **`Document`'s not restored reasons** is its node navigable[p1036]'s active session history entry[p1035]'s document state[p1053]'s not restored reasons[p1055], if `Document`[p135]'s node navigable[p1036] is a top-level traversable[p1037]; otherwise null.

7.3 Infrastructure for sequences of documents §[p1035]

This standard contains several related concepts for grouping sequences of documents. As a brief, non-normative summary:

- Navigables[p1035] are a user-facing representation of a sequence of documents, i.e., they represent something that can be navigated between documents. Typical examples are tabs or windows in a web browser, or `iframe`[p404]s, or `frame`[p1537]s in a `frameset`[p1537].
- Traversable navigables[p1037] are a special type of navigable which control the session history of themselves and of their descendant navigables. That is, in addition to their own series of documents, they represent a tree of further series of documents, plus the ability to linearly traverse back and forward through a flattened view of this tree.
- Browsing contexts[p1046] are a developer-facing representation of a series of documents. They correspond 1:1 with `WindowProxy`[p977] objects. Each navigable can present a series of browsing contexts, with switches[p947] between those browsing contexts occurring under certain well-defined circumstances.

Most of this standard works in the language of navigables, but certain APIs expose the existence of browsing context switches, and so some parts of the standard need to work in terms of browsing contexts.

7.3.1 Navigables §[p1035]

A **navigable** presents a `Document`[p135] to the user via its active session history entry[p1035]. Each navigable has:

- An **id**, a new unique internal value[p100].
- A **parent**, a navigable[p1035] or null.
- A **current session history entry**, a session history entry[p1053].

  This can only be modified within the session history traversal queue[p1037] of the parent traversable navigable[p1037].
- An **active session history entry**, a session history entry[p1053].

This can only be modified from the event loop of the active session history entry[p1035]'s document[p1053].

- An **is closing** boolean, initially false.

  Note

  This is only ever set to true for top-level traversable navigables[p1037].

- An **is delaying `load` events** boolean, initially false.

  Note

  This is only ever set to true in cases where the navigable's parent[p1035] is non-null.

- An **allowed to perform a navigation or history update** algorithm, which is implementation-defined and returns either allowed or blocked.

  Note

  For example, this can return blocked if invoked too many times times within a certain timespan.

The current session history entry[p1035] and the active session history entry[p1035] are usually the same, but they get out of sync when:

- Synchronous navigations are performed. This causes the active session history entry[p1035] to temporarily step ahead of the current session history entry[p1035].
- A non-displayable, non-error response is received when applying the history step[p1091]. This updates the current session history entry[p1035] but leaves the active session history entry[p1035] as-is.

A navigable[p1035]'s **active document** is its active session history entry[p1035]'s document[p1053].

Note

This can be safely read from within the session history traversal queue[p1037] of the navigable's top-level traversable[p1037]. Although a navigable[p1035]'s active history entry[p1035] can change synchronously, the new entry will always have the same `Document`[p135].

A navigable[p1035]'s **active browsing context** is its active document[p1036]'s browsing context[p1046]. If this navigable[p1035] is a traversable navigable[p1037], then its active browsing context[p1036] will be a top-level browsing context[p1049].

A navigable[p1035]'s **active `WindowProxy`** is its active browsing context[p1036]'s associated `WindowProxy`[p977].

A navigable[p1035]'s **active window** is its active `WindowProxy`[p1036]'s [[Window]][p977].

Note

This will always equal the navigable's active document[p1036]'s relevant global object[p1151]; this is kept in sync by the make active[p1101] algorithm.

A navigable[p1035]'s **target name** is its active session history entry[p1035]'s document state[p1053]'s navigable target name[p1055].

To get the **node navigable** of a node *node*, return the navigable[p1035] whose active document[p1036] is *node*'s node document, or null if there is no such navigable[p1035].

To **initialize the navigable** navigable[p1035] *navigable*, given a document state[p1054] *documentState* and an optional navigable[p1035]-or-null *parent* (default null):

1. Assert: *documentState*'s document[p1054] is non-null.
2. Let *entry* be a new session history entry[p1053], with

   URL[p1053]
   documentState's document[p1054]'s URL

   document state[p1053]
   documentState

Note

The caller of this algorithm is responsible for initializing entry's *step*[p1053]*; it will be left as* "pending" *until that is complete.*

3. Set *navigable*'s current session history entry[p1035] to *entry*.
4. Set *navigable*'s active session history entry[p1035] to *entry*.
5. Set *navigable*'s parent[p1035] to *parent*.
6. Set the initial visibility state[p865] of *documentState*'s document[p1054] to *navigable*'s traversable navigable[p1037]'s system visibility state[p1037].

7.3.1.1 Traversable navigables §[p1037]

A **traversable navigable** is a navigable[p1035] that also controls which session history entry[p1053] should be the current session history entry[p1035] and active session history entry[p1035] for itself and its descendant navigables[p1035].

In addition to the properties of a navigable[p1035], a traversable navigable[p1037] has:

- A **current session history step**, a number, initially 0.
- **Session history entries**, a list of session history entries[p1053], initially a new list.
- A **session history traversal queue**, a session history traversal parallel queue[p1056], the result of starting a new session history traversal parallel queue[p1056].
- A **running nested apply history step** boolean, initially false.
- A **system visibility state**, which is either "hidden" or "visible".

  See the page visibility[p864] section for the requirements on this item.
- An **is created by web content** boolean, initially false.

To get the **traversable navigable** of a navigable[p1035] *inputNavigable*:

1. Let *navigable* be *inputNavigable*.
2. While *navigable* is not a traversable navigable[p1037], set *navigable* to *navigable*'s parent[p1035].
3. Return *navigable*.

7.3.1.2 Top-level traversables §[p1037]

A **top-level traversable** is a traversable navigable[p1037] with a null parent[p1035].

Note

Currently, all traversable navigables[p1037] *are top-level traversables*[p1037]*. Future proposals envision introducing non-top-level traversables.*

A user agent holds a **top-level traversable set** (a set of top-level traversables[p1037]). These are typically presented to the user in the form of browser windows or browser tabs.

To get the **top-level traversable** of a navigable[p1035] *inputNavigable*:

1. Let *navigable* be *inputNavigable*.
2. While *navigable*'s parent[p1035] is not null, set *navigable* to *navigable*'s parent[p1035].
3. Return *navigable*.

To **create a new top-level traversable** given a browsing context[p1046]-or-null *opener*, a string *targetName*, and an optional navigable[p1035] *openerNavigableForWebDriver*:

1. Let *document* be null.
2. If *opener* is null, then set *document* to the second return value of creating a new top-level browsing context and document[p1048].
3. Otherwise, set *document* to the second return value of creating a new auxiliary browsing context and document[p1048] given *opener*.
4. Let *documentState* be a new document state[p1054], with

   document[p1054]
   document
   initiator origin[p1054]
   null if *opener* is null; otherwise, *document*'s origin
   origin[p1054]
   document's origin
   navigable target name[p1055]
   targetName
   about base URL[p1054]
   document's about base URL[p136]
5. Let *traversable* be a new traversable navigable[p1037].
6. Initialize the navigable[p1036] *traversable* given *documentState*.
7. Let *initialHistoryEntry* be *traversable*'s active session history entry[p1035].
8. Set *initialHistoryEntry*'s step[p1053] to 0.
9. Append *initialHistoryEntry* to *traversable*'s session history entries[p1037].
10. If *opener* is non-null, then legacy-clone a traversable storage shed given *opener*'s top-level traversable[p1046] and *traversable*. [STORAGE][p1586]
11. Append *traversable* to the user agent's top-level traversable set[p1037].
12. Invoke WebDriver BiDi navigable created with *traversable* and *openerNavigableForWebDriver*.
13. Return *traversable*.

To **create a fresh top-level traversable** given a URL *initialNavigationURL* and an optional POST resource[p1055]-or-null *initialNavigationPostResource* (default null):

1. Let *traversable* be the result of creating a new top-level traversable[p1037] given null and the empty string.
2. Navigate[p1063] *traversable* to *initialNavigationURL* using *traversable*'s active document[p1036], with *documentResource*[p1063] set to *initialNavigationPostResource*.

   Note

   We treat these initial navigations as traversable *navigating itself, which will ensure all relevant security checks pass.*
3. Return *traversable*.

7.3.1.3 Child navigables §[p1038]

Certain elements (for example, `iframe`[p404] elements) can present a navigable[p1035] to the user. These elements are called **navigable containers**.

Each navigable container[p1038] has a **content navigable**, which is either a navigable[p1035] or null. It is initially null.

The **container** of a navigable[p1035] *navigable* is the navigable container[p1038] whose content navigable[p1038] is *navigable*, or null if there is no such element.

The **container document** of a navigable[p1035] *navigable* is the result of running these steps:

1. If *navigable*'s container[p1038] is null, then return null.

2. Return *navigable*'s container[p1038]'s node document.

Note

This is equal to navigable's *container*[p1038]'s *shadow-including root as* navigable's *container*[p1038] *has to be connected.*

The **container document** of a `Document`[p135] *document* is the result of running these steps:

1. If *document*'s node navigable[p1036] is null, then return null.
2. Return *document*'s node navigable[p1036]'s container document[p1038].

A navigable[p1035] *navigable* is a **child navigable** of another navigable *potentialParent* when *navigable*'s parent[p1035] is *potentialParent*. We can also just say that a navigable[p1035] "is a child navigable[p1039]", which means that its parent[p1035] is non-null.

Note

All child navigables[p1039] *are the content navigable*[p1038] *of their container*[p1038].

The **content document** of a navigable container[p1038] *container* is the result of running these steps:

1. If *container*'s content navigable[p1038] is null, then return null.
2. Let *document* be *container*'s content navigable[p1038]'s active document[p1036].
3. If *document*'s origin and *container*'s node document's origin are not same origin-domain[p940], then return null.
4. Return *document*.

The **content window** of a navigable container[p1038] *container* is the result of running these steps:

1. If *container*'s content navigable[p1038] is null, then return null.
2. Return *container*'s content navigable[p1038]'s active `WindowProxy`[p1036]'s object.

To **create a new child navigable**, given an element *element*:

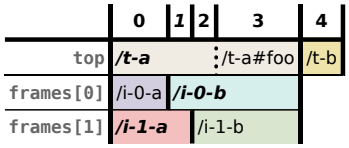
1. Let *parentNavigable* be *element*'s node navigable[p1036].
2. Let *group* be *element*'s node document's browsing context[p1046]'s top-level browsing context[p1049]'s group[p1050].
3. Let *browsingContext* and *document* be the result of creating a new browsing context and document[p1047] given *element*'s node document, *element*, and *group*.
4. Let *targetName* be null.
5. If *element* has a `name` content attribute, then set *targetName* to the value of that attribute.
6. Let *documentState* be a new document state[p1054], with

   document[p1054]
   document
   initiator origin[p1054]
   document's origin
   origin[p1054]
   document's origin
   navigable target name[p1055]
   targetName
   about base URL[p1054]
   document's about base URL[p136]
7. Let *navigable* be a new navigable[p1035].
8. Initialize the navigable[p1036] *navigable* given *documentState* and *parentNavigable*.
9. Set *element*'s content navigable[p1038] to *navigable*.
10. Let *historyEntry* be *navigable*'s active session history entry[p1035].

11. Let *traversable* be *parentNavigable*'s traversable navigable[p1037].
12. Append the following session history traversal steps[p1056] to *traversable*:
    1. Let *parentDocState* be *parentNavigable*'s active session history entry[p1035]'s document state[p1053].
    2. Let *parentNavigableEntries* be the result of getting session history entries[p1058] for *parentNavigable*.
    3. Let *targetStepSHE* be the first session history entry[p1053] in *parentNavigableEntries* whose document state[p1053] equals *parentDocState*.
    4. Set *historyEntry*'s step[p1053] to *targetStepSHE*'s step[p1053].
    5. Let *nestedHistory* be a new nested history[p1055] whose id[p1055] is *navigable*'s id[p1035] and entries list[p1055] is « *historyEntry* ».
    6. Append *nestedHistory* to *parentDocState*'s nested histories[p1054].
    7. Update for navigable creation/destruction[p1090] given *traversable*.
13. Invoke WebDriver BiDi navigable created with *traversable*.

7.3.1.4 Jake diagrams §[p10 40]

A useful method for visualizing sequences of documents, and in particular navigables[p1035] and their session history entries[p1053], is the **Jake diagram**. A typical Jake diagram is the following:



Here, each numbered column denotes a possible value for the traversable's session history step[p1037]. Each labeled row depicts a navigable[p1035], as it transitions between different URLs and documents. The first, labeled `top`, being the top-level traversable[p1037], and the others being child navigables[p1039]. The documents are given by the background color of each cell, with a new background color indicating a new document in that navigable[p1035]. The URLs are given by the text content of the cells; usually they are given as relative URLs for brevity, unless a cross-origin case is specifically under investigation. A given navigable might not exist at a given step, in which case the corresponding cells are empty. The bold-italic step number depicts the current session history step[p1037] of the traversable, and all cells with bold-italic URLs represent the current session history entry[p1035] for that row's navigable.

Thus, the above Jake diagram depicts the following sequence of events:

0. A top-level traversable[p1037] is created, starting at the URL `/t-a`, with two child navigables[p1039] starting at `/i-0-a` and `/i-1-a` respectively.
1. The first child navigable is navigated[p1063] to another document, with URL `/i-0-b`.
2. The second child navigable is navigated[p1063] to another document, with URL `/i-1-b`.
3. The top-level traversable is navigated[p1063] to the *same* document, updating its URL to `/t-a#foo`.
4. The top-level traversable is navigated[p1063] to another document, with URL `/t-b`. (Notice how this document, of course, does not carry over the old document's child navigables.)
5. The traversable was traversed by a delta[p1077] of −3, back to step 1.

Jake diagrams[p1040] are a powerful tool for visualizing the interactions of multiple navigables, navigations, and traversals. They cannot capture every possible interaction — for example, they only work with a single level of nesting — but we will have ocassion to use them to illustrate several complex situations throughout this standard.

Note

Jake diagrams[p1040] are named after their creator, the inimitable Jake Archibald.

7.3.1.5 Related navigable collections

It is often helpful in this standard's algorithms to look at collections of navigables starting at a given Document. This section contains a curated set of algorithms for collecting those navigables.

Note

The return values of these algorithms are ordered so that parents appears before their children. Callers rely on this ordering.

Note

Starting with a Document, rather than a navigable, is generally better because it makes the caller cognizant of whether they are starting with a fully active Document or not. Although non-fully active Documents do have ancestor and descendant navigables, they often behave as if they don't (e.g., in the `window.parent` getter).

The **ancestor navigables** of a Document *document* are given by these steps:

1. Let *navigable* be *document*'s node navigable's parent.
2. Let *ancestors* be an empty list.
3. While *navigable* is not null:
    1. Prepend *navigable* to *ancestors*.
    2. Set *navigable* to *navigable*'s parent.
4. Return *ancestors*.

The **inclusive ancestor navigables** of a Document *document* are given by these steps:

1. Let *navigables* be *document*'s ancestor navigables.
2. Append *document*'s node navigable to *navigables*.
3. Return *navigables*.

The **descendant navigables** of a Document *document* are given by these steps:

1. Let *navigables* be new list.
2. Let *navigableContainers* be a list of all shadow-including descendants of *document* that are navigable containers, in shadow-including tree order.
3. For each *navigableContainer* of *navigableContainers*:
    1. If *navigableContainer*'s content navigable is null, then continue.
    2. Extend *navigables* with *navigableContainer*'s content navigable's active document's inclusive descendant navigables.
4. Return *navigables*.

The **inclusive descendant navigables** of a Document *document* are given by these steps:

1. Let *navigables* be « *document*'s node navigable ».
2. Extend *navigables* with *document*'s descendant navigables.
3. Return *navigables*.

Note

These descendant-collecting algorithms are described as looking at the DOM tree of descendant Document objects. In reality, this is often not feasible since the DOM tree can be in another process from the caller of the algorithm. Instead, implementations generally replicate the appropriate trees across processes.

The **document-tree child navigables** of a Document *document* are given by these steps:

1. If *document*'s node navigable[p1036] is null, then return the empty list.
2. Let *navigables* be new list.
3. Let *navigableContainers* be a list of all descendants of *document* that are navigable containers[p1038], in tree order.
4. For each *navigableContainer* of *navigableContainers*:
    1. If *navigableContainer*'s content navigable[p1038] is null, then continue.
    2. Append *navigableContainer*'s content navigable[p1038] to *navigables*.
5. Return *navigables*.

7.3.1.6 Navigable destruction §[p1042]

To **destroy a child navigable** given a navigable container[p1038] *container*:

1. Let *navigable* be *container*'s content navigable[p1038].
2. If *navigable* is null, then return.
3. Set *container*'s content navigable[p1038] to null.
4. Inform the navigation API about child navigable destruction[p1011] given *navigable*.
5. Destroy a document and its descendants[p1117] given *navigable*'s active document[p1036].
6. Let *parentDocState* be *container*'s node navigable[p1036]'s active session history entry[p1035]'s document state[p1053].
7. Remove the nested history[p1055] from *parentDocState*'s nested histories[p1054] whose id[p1055] equals *navigable*'s id[p1035].
8. Let *traversable* be *container*'s node navigable[p1036]'s traversable navigable[p1037].
9. Append the following session history traversal steps[p1056] to *traversable*:
    1. Update for navigable creation/destruction[p1090] given *traversable*.
10. Invoke WebDriver BiDi navigable destroyed with *navigable*.

To **destroy** a top-level traversable[p1037] *traversable*:

1. Let *browsingContext* be *traversable*'s active browsing context[p1036].
2. For each *historyEntry* in *traversable*'s session history entries[p1037] in what order?:
    1. Let *document* be *historyEntry*'s document[p1053].
    2. If *document* is not null, then destroy a document and its descendants[p1117] given *document*.
3. Remove[p1051] *browsingContext*.
4. Remove *traversable* from the user interface (e.g., close or hide its tab in a tabbed browser).
5. Remove *traversable* from the user agent's top-level traversable set[p1037].
6. Invoke WebDriver BiDi navigable destroyed with *traversable*.

User agents may destroy a top-level traversable[p1042] at any time (typically, in response to user requests[p1137]).

To **close** a top-level traversable[p1037] *traversable*:

1. If *traversable*'s is closing[p1036] is true, then return.
2. Definitely close[p1042] *traversable*.

To **definitely close** a top-level traversable[p1037] *traversable*:

1. Let *toUnload* be *traversable*'s active document[p1036]'s inclusive descendant navigables[p1041].

2. If the result of checking if unloading is canceled[p1074] for *toUnload* is not "`continue`", then return.

3. Append the following session history traversal steps[p1056] to *traversable*:

    1. Let *afterAllUnloads* be an algorithm step which destroys[p1042] *traversable*.

    2. Unload a document and its descendants[p1115] given *traversable*'s active document[p1036], null, and *afterAllUnloads*.

Note

The close[p1042] vs. definitely close[p1042] separation allows other specifications to call close[p1042] and have it be a no-op if the top-level traversable is already closing due to JavaScript code calling `window.close()`[p971].

7.3.1.7 Navigable target names §[p10 43]

Navigables[p1035] can be given target names[p1036], which are strings allowing certain APIs (such as `window.open()`[p969] or the `a`[p267] element's `target`[p314] attribute) to target navigations[p1063] at that navigable.

A **valid navigable target name** is any string with at least one character that does not contain both an ASCII tab or newline and a U+003C (<), and it does not start with a U+005F (_). (Names starting with a U+005F (_) are reserved for special keywords.)

A **valid navigable target name or keyword** is any string that is either a valid navigable target name[p1043] 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 navigable[p1035] that the link or script is in, "parent" means the parent[p1035] of the navigable[p1035] that the link or script is in, "top" means the top-level traversable[p1037] of the navigable[p1035] that the link or script is in, "new" means a new traversable navigable[p1037] with a null parent[p1035] (which may use an auxiliary browsing context[p1046], 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`[p958]" keyword is also specified on the `sandbox`[p409] attribute (or if the user overrode the sandboxing), and the same as "none" otherwise.

Keyword	Ordinary effect	Effect in an `iframe`[p404] with...	
		`sandbox=""`	`sandbox="allow-top-navigation"`
none specified, for links and form submissions	current	current	current
empty string	current	current	current
`_blank`	new	maybe new	maybe new
`_self`	current	current	current
`_parent` if there isn't a parent	current	current	current
`_parent` if parent is also top	parent/top	none	parent/top
`_parent` if there is one and it's not top	parent	none	none
`_top` if top is current	current	current	current
`_top` if top is not current	top	none	top
name that doesn't exist	new	maybe new	maybe new
name that exists and is a descendant	specified descendant	specified descendant	specified descendant
name that exists and is current	current	current	current
name that exists and is an ancestor that is top	specified ancestor	none	specified ancestor/top
name that exists and is an ancestor that is not top	specified ancestor	none	none
other name that exists with common top	specified	none	none
name that exists with different top, if familiar[p1050] and one permitted sandboxed navigator[p957]	specified	specified	specified
name that exists with different top, if familiar[p1050] but not one permitted sandboxed navigator[p957]	specified	none	none
name that exists with different top, not familiar[p1050]	new	maybe new	maybe new

Most of the restrictions on sandboxed browsing contexts are applied by other algorithms, e.g. the navigation[p1063] algorithm, not the rules for choosing a navigable[p1044] given below.

To **find a navigable by target name** given a string *name* and a navigable[p1035] *currentNavigable*:

1. Let *currentDocument* be *currentNavigable*'s active document[p1036].
2. Let *sourceSnapshotParams* be the result of snapshotting source snapshot params[p1060] given *currentDocument*.
3. Let *subtreesToSearch* be an implementation-defined choice of one of the following:
    - « *currentNavigable*'s traversable navigable[p1037], *currentNavigable* »
    - the inclusive ancestor navigables[p1041] of *currentDocument*

    Issue #10848 tracks settling on one of these two possibilities, to achieve interoperability.
4. For each *subtreeToSearch* of *subtreesToSearch*, in reverse order:
    1. Let *documentToSearch* be *subtreeToSearch*'s active document[p1036].
    2. For each *navigable* of the inclusive descendant navigables[p1041] of *documentToSearch*:
        1. If *currentNavigable* is not allowed by sandboxing to navigate[p1074] *navigable* given *sourceSnapshotParams*, then optionally continue.

            Issue #10849 tracks making this check required, to achieve interoperability.
        2. If *navigable*'s target name[p1036] is *name*, then return *navigable*.
5. Let *currentTopLevelBrowsingContext* be *currentNavigable*'s active browsing context[p1036]'s top-level browsing context[p1049].
6. Let *group* be *currentTopLevelBrowsingContext*'s group[p1050].
7. For each *topLevelBrowsingContext* of *group*'s browsing context set[p1050], in an implementation-defined order (the user agent should pick a consistent ordering, such as the most recently opened, most recently focused, or more closely related):

    Issue #10850 tracks picking a specific ordering, to achieve interoperability.
    1. If *currentTopLevelBrowsingContext* is *topLevelBrowsingContext*, then continue.
    2. Let *documentToSearch* be *topLevelBrowsingContext*'s active document[p1046].
    3. For each *navigable* of the inclusive descendant navigables[p1041] of *documentToSearch*:
        1. If *currentNavigable*'s active browsing context[p1036] is not familiar with[p1050] *navigable*'s active browsing context[p1036], then continue.
        2. If *currentNavigable* is not allowed by sandboxing to navigate[p1074] *navigable* given *sourceSnapshotParams*, then optionally continue.

            Issue #10849 tracks making this check required, to achieve interoperability.
        3. If *navigable*'s target name[p1036] is *name*, then return *navigable*.
8. Return null.

The rules for choosing a navigable, given a string *name*, a navigable[p1035] *currentNavigable*, and a boolean *noopener* are as follows:

1. Let *chosen* be null.
2. Let *windowType* be "`existing or none`".
3. Let *sandboxingFlagSet* be *currentNavigable*'s active document[p1036]'s active sandboxing flag set[p959].
4. If *name* is the empty string or an ASCII case-insensitive match for "`_self`", then set *chosen* to *currentNavigable*.
5. Otherwise, if *name* is an ASCII case-insensitive match for "`_parent`", set *chosen* to *currentNavigable*'s parent[p1035], if any, and *currentNavigable* otherwise.
6. Otherwise, if *name* is an ASCII case-insensitive match for "`_top`", set *chosen* to *currentNavigable*'s traversable navigable[p1037].

7. Otherwise, if *name* is not an ASCII case-insensitive match for "`_blank`", then set *chosen* to the result of finding a navigable by target name[p1044] given *name* and *currentNavigable*.

8. If *chosen* is null, then a new top-level traversable[p1037] 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 *currentNavigable*'s active window[p1036] does not have transient activation[p868] 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[p957] 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 top-level traversable[p1037]

   1. Consume user activation[p869] of *currentNavigable*'s active window[p1036].
   2. Set *windowType* to "`new and unrestricted`".
   3. Let *currentDocument* be *currentNavigable*'s active document[p1036].
   4. If *currentDocument*'s opener policy[p136]'s value[p945] is "`same-origin`[p945]" or "`same-origin-plus-COEP`[p945]", and *currentDocument*'s origin is not same origin[p940] with *currentDocument*'s relevant settings object[p1151]'s top-level origin[p1144]:
      1. Set *noopener* to true.
      2. Set *name* to "`_blank`".
      3. Set *windowType* to "`new with no opener`".

      Note

      In the presence of an opener policy[p945], nested documents that are cross-origin with their top-level browsing context's active document always set noopener *to true.*
   5. Let *targetName* be the empty string.
   6. If *name* is not an ASCII case-insensitive match for "`_blank`", then set *targetName* to *name*.
   7. If *noopener* is true, then set *chosen* to the result of creating a new top-level traversable[p1037] given null, *targetName*, and *currentNavigable*.
   8. Otherwise:
      1. Set *chosen* to the result of creating a new top-level traversable[p1037] given *currentNavigable*'s active browsing context[p1036], *targetName*, and *currentNavigable*.
      2. If *sandboxingFlagSet*'s sandboxed navigation browsing context flag[p957] is set, then set *chosen*'s active browsing context[p1036]'s one permitted sandboxed navigator[p957] to *currentNavigable*'s active browsing context[p1036].
   9. If *sandboxingFlagSet*'s sandbox propagates to auxiliary browsing contexts flag[p958] is set, then all the flags that are set in *sandboxingFlagSet* must be set in *chosen*'s active browsing context[p1036]'s popup sandboxing flag set[p959].
   10. Set *chosen*'s is created by web content[p1037] to true.

   Note

   If the newly created navigable[p1035] chosen *is immediately navigated[p1063], then the navigation will be done as a "`replace`[p1062]" navigation.*

   ↪ If the user agent has been configured such that in this instance it will choose *currentNavigable*

   Set *chosen* to *currentNavigable*.

↪ **If the user agent has been configured such that in this instance it will not find a navigable**

Do nothing.

Note

User agents are encouraged to provide a way for users to configure the user agent to always choose currentNavigable.

9. Return *chosen* and *windowType*.

7.3.2 Browsing contexts §p1046

A **browsing context** is a programmatic representation of a series of documents, multiple of which can live within a single navigable[p1035]. Each browsing context[p1046] has a corresponding `WindowProxy`[p977] object, as well as the following:

- An **active document**, a `Document`[p135] or null, initially null.
- An **opener browsing context**, a browsing context[p1046] or null, initially null.
- An **opener origin at creation**, an origin[p939] or null, initially null.
- An **is popup** boolean, initially false.

  Note

  The only mandatory impact in this specification of is popup[p1046] is on the `visible`[p975] getter of the relevant `BarProp`[p975] objects. However, user agents might also use it for user interface considerations[p1137].

- An **is auxiliary** boolean, initially false.
- An **initial URL**, a URL or null, initially null.
- A **virtual browsing context group ID** integer, initially 0. This is used by opener policy reporting[p946], to keep track of the browsing context group switches that would have happened if the report-only policy had been enforced.

A browsing context[p1046]'s **active window** is its `WindowProxy`[p977] object's [[Window]][p977] internal slot value.

A browsing context[p1046]'s **top-level traversable** is its active document[p1046]'s node navigable[p1036]'s top-level traversable[p1037].

A browsing context[p1046] whose is auxiliary[p1046] is true is known as an **auxiliary browsing context**. Auxiliary browsing contexts are always top-level browsing contexts[p1049].

It's unclear whether a separate is auxiliary[p1046] concept is necessary. In issue #5680, it is indicated that we may be able to simplify this by using whether or not the opener browsing context[p1046] is null.

⚠Warning!

Modern specifications should avoid using the browsing context[p1046] concept in most cases, unless they are dealing with the subtleties of browsing context group switches[p947] and agent cluster allocation[p1050]. Instead, the `Document`[p135] and navigable[p1035] concepts are usually more appropriate.

A **`Document`'s browsing context** is a browsing context[p1046] or null, initially null.

Note

A `Document`[p135] does not necessarily have a non-null browsing context[p1046]. In particular, data mining tools are likely to never instantiate browsing contexts. A `Document`[p135] created using an API such as `createDocument()` never has a non-null browsing context[p1046]. And the `Document`[p135] originally created for an `iframe`[p404] element, which has since been removed from the document[p47], has no associated browsing context, since that browsing context was nulled out[p1116].

Note

In general, there is a 1-to-1 mapping from the `Window`[p965] object to the `Document`[p135] object, as long as the `Document`[p135] object has a non-null browsing context[p1046]. There is one exception. A `Window`[p965] can be reused for the presentation of a second `Document`[p135]

in the same browsing context[p1046], such that the mapping is then 1-to-2. This occurs when a browsing context[p1046] is navigated[p1063] from the initial about:blank[p136] Document[p135] to another, which will be done with replacement[p1062].

7.3.2.1 Creating browsing contexts §[p1047]

To **create a new browsing context and document**, given null or a Document[p135] object *creator*, null or an element *embedder*, and a browsing context group[p1050] *group*:

1. Let *browsingContext* be a new browsing context[p1046].
2. Let *unsafeContextCreationTime* be the unsafe shared current time.
3. Let *creatorOrigin* be null.
4. Let *creatorBaseURL* be null.
5. If *creator* is non-null:
    1. Set *creatorOrigin* to *creator*'s origin.
    2. Set *creatorBaseURL* to *creator*'s document base URL[p101].
    3. Set *browsingContext*'s virtual browsing context group ID[p1046] to *creator*'s browsing context[p1046]'s top-level browsing context[p1049]'s virtual browsing context group ID[p1046].
6. Let *sandboxFlags* be the result of determining the creation sandboxing flags[p960] given *browsingContext* and *embedder*.
7. Let *origin* be the result of determining the origin[p1049] given about:blank[p54], *sandboxFlags*, and *creatorOrigin*.
8. Let *permissionsPolicy* be the result of creating a permissions policy given *embedder* and *origin*. [PERMISSIONSPOLICY][p1585]
9. Let *agent* be the result of obtaining a similar-origin window agent[p1141] given *origin*, *group*, and false.
10. Let *realm execution context* be the result of creating a new realm[p1145] given *agent* and the following customizations:
    - For the global object, create a new Window[p965] object.
    - For the global **this** binding, use *browsingContext*'s WindowProxy[p977] object.
11. Let *topLevelCreationURL* be about:blank[p54] if *embedder* is null; otherwise *embedder*'s relevant settings object[p1151]'s top-level creation URL[p1144].
12. Let *topLevelOrigin* be *origin* if *embedder* is null; otherwise *embedder*'s relevant settings object[p1151]'s top-level origin[p1144].
13. Set up a window environment settings object[p976] with about:blank[p54], *realm execution context*, null, *topLevelCreationURL*, and *topLevelOrigin*.
14. Let *loadTimingInfo* be a new document load timing info[p142] with its navigation start time[p142] set to the result of calling coarsen time with *unsafeContextCreationTime* and the new environment settings object[p1144]'s cross-origin isolated capability[p1144].
15. Let *document* be a new Document[p135], with:

    type
    "html"
    content type
    "text/html[p1546]"
    mode
    "quirks"
    origin
    origin
    browsing context[p1046]
    browsingContext
    permissions policy[p136]
    permissionsPolicy

active sandboxing flag set[p959]
sandboxFlags
load timing info[p141]
loadTimingInfo
is initial about:blank[p136]
true
about base URL[p136]
creatorBaseURL
allow declarative shadow roots
true
custom element registry
a new CustomElementRegistry[p799] object

16. Let *iframeReferrerPolicy* be the result of determining the iframe element referrer policy[p960] given *embedder*.
17. Set *document*'s internal ancestor origin objects list[p137] to the result of running the internal ancestor origin objects list creation steps[p137] given *document* and *iframeReferrerPolicy*.
18. Set *document*'s ancestor origins list[p138] to the result of running the ancestor origins list creation steps[p138] given *document*.
19. If *creator* is non-null:
    1. Set *document*'s referrer[p135] to the serialization of *creator*'s URL.
    2. Set *document*'s policy container[p136] to a clone[p960] of *creator*'s policy container[p136].
    3. If *creator*'s origin is same origin[p940] with *creator*'s relevant settings object[p1151]'s top-level origin[p1144], then set *document*'s opener policy[p136] to *creator*'s browsing context[p1046]'s top-level browsing context[p1049]'s active document[p1046]'s opener policy[p136].
20. Assert: *document*'s URL and *document*'s relevant settings object[p1151]'s creation URL[p1143] are about:blank[p54].
21. Mark *document* as ready for post-load tasks[p1459].
22. Populate with html/head/body[p1109] given *document*.
23. Make active[p1101] *document*.
24. Completely finish loading[p1114] *document*.
25. Return *browsingContext* and *document*.

To **create a new top-level browsing context and document**:

1. Let *group* and *document* be the result of creating a new browsing context group and document[p1050].
2. Return *group*'s browsing context set[p1050][0] and *document*.

To **create a new auxiliary browsing context and document**, given a browsing context[p1046] *opener*:

1. Let *openerTopLevelBrowsingContext* be *opener*'s top-level traversable[p1046]'s active browsing context[p1036].
2. Let *group* be *openerTopLevelBrowsingContext*'s group[p1050].
3. Assert: *group* is non-null, as navigating[p1063] invokes this directly.
4. Let *browsingContext* and *document* be the result of creating a new browsing context and document[p1047] with *opener*'s active document[p1036], null, and *group*.
5. Set *browsingContext*'s is auxiliary[p1046] to true.
6. Append[p1051] *browsingContext* to *group*.
7. Set *browsingContext*'s opener browsing context[p1046] to *opener*.
8. Set *browsingContext*'s virtual browsing context group ID[p1046] to *openerTopLevelBrowsingContext*'s virtual browsing context group ID[p1046].
9. Set *browsingContext*'s opener origin at creation[p1046] to *opener*'s active document[p1036]'s origin.

10. Return *browsingContext* and *document*.

To **determine the origin**, given a URL *url*, a sandboxing flag set[p957] *sandboxFlags*, and an origin[p939]-or-null *sourceOrigin*:

1. If *sandboxFlags* has its sandboxed origin browsing context flag[p957] set, then return a new opaque origin[p939].
2. If *url* is null, then return a new opaque origin[p939].
3. If *url* is `about:srcdoc`[p100]:
    1. Assert: *sourceOrigin* is non-null.
    2. Return *sourceOrigin*.
4. If *url* matches `about:blank`[p100] and *sourceOrigin* is non-null, then return *sourceOrigin*.
5. Return *url*'s origin.

Note

The cases that return sourceOrigin *result in two* `Document`[p135]*s that end up with the same underlying origin, meaning that* `document.domain`[p942] *affects both.*

7.3.2.2 Related browsing contexts §[p10 49]

A browsing context[p1046] *potentialDescendant* is said to be an **ancestor** of a browsing context *potentialAncestor* if the following algorithm returns true:

1. Let *potentialDescendantDocument* be *potentialDescendant*'s active document[p1046].
2. If *potentialDescendantDocument* is not fully active[p1051], then return false.
3. Let *ancestorBCs* be the list obtained by taking the browsing context[p1046] of the active document[p1036] of each member of *potentialDescendantDocument*'s ancestor navigables[p1041].
4. If *ancestorBCs* contains *potentialAncestor*, then return true.
5. Return false.

A **top-level browsing context** is a browsing context[p1046] whose active document[p1046]'s node navigable[p1036] is a traversable navigable[p1037].

Note

It is not *required to be a top-level traversable*[p1037].

The **top-level browsing context** of a browsing context[p1046] *start* is the result of the following algorithm:

1. If *start*'s active document[p1046] is not fully active[p1051], then return null.
2. Let *navigable* be *start*'s active document[p1046]'s node navigable[p1036].
3. While *navigable*'s parent[p1035] is not null, set *navigable* to *navigable*'s parent[p1035].
4. Return *navigable*'s active browsing context[p1036].

⚠Warning!

The terms ancestor browsing context*[p1049] *and top-level browsing context*[p1049] *are rarely useful, since browsing contexts*[p1046] *in general are usually the inappropriate specification concept to use*[p1046]*. Note in particular that when a browsing context*[p1046]*'s active document*[p1046] *is not fully active*[p1051]*, it never counts as an ancestor or top-level browsing context, and as such these concepts are not useful when bfcache*[p1054] *is in play.

Instead, use concepts such as the ancestor navigables*[p1041] *collection, the parent navigable*[p1035]*, or a navigable's top-level traversable*[p1037]*.

A browsing context[p1046] *A* is **familiar with** a second browsing context[p1046] *B* if the following algorithm returns true:

1. If *A*'s active document[p1046]'s origin is same origin[p940] with *B*'s active document[p1046]'s origin, then return true.
2. If *A*'s top-level browsing context[p1049] is *B*, then return true.
3. If *B* is an auxiliary browsing context[p1046] and *A* is familiar with[p1050] *B*'s opener browsing context[p1046], then return true.
4. If there exists an ancestor browsing context[p1049] of *B* whose active document[p1046] has the same[p940] origin as the active document[p1046] of *A*, then return true.

   Note

   This includes the case where A *is an ancestor browsing context[p1049] of* B.

5. Return false.

7.3.2.3 Groupings of browsing contexts §[p1050]

A top-level browsing context[p1049] has an associated **group** (null or a browsing context group[p1050]). It is initially null.

A user agent holds a **browsing context group set** (a set of browsing context groups[p1050]).

A **browsing context group** holds a **browsing context set** (a set of top-level browsing contexts[p1049]).

Note

A top-level browsing context[p1049] is added to the group[p1050] when the group is created[p1050]. All subsequent top-level browsing contexts[p1049] added to the group[p1050] will be auxiliary browsing contexts[p1046].

A browsing context group[p1050] has an associated **agent cluster map** (a weak map of agent cluster keys[p1141] 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[p1050] has an associated **historical agent cluster key map**, which is a map of origins[p939] to agent cluster keys[p1141]. This map is used to ensure the consistency of the origin-keyed agent clusters[p944] feature by recording what agent cluster keys were previously used for a given origin.

Note

The historical agent cluster key map[p1050] only ever gains entries over the lifetime of the browsing context group.

A browsing context group[p1050] has a **cross-origin isolation mode**, which is a cross-origin isolation mode[p1050]. It is initially "`none`[p1050]".

A **cross-origin isolation mode** is one of three possible values: "**`none`**", "**`logical`**", or "**`concrete`**".

Note

"`logical`[p1050]" *and* "`concrete`[p1050]" *are similar. They are both used for browsing context groups[p1050] where:*

- *every top-level* `Document`[p135] *has* `` `Cross-Origin-Opener-Policy`[p946]: `same-origin`[p945] ``*, and*
- *every* `Document`[p135] *has a* `` `Cross-Origin-Embedder-Policy`[p955] `` *header whose value is compatible with cross-origin isolation[p955].*

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[p1144]. As a result, only "`concrete`[p1050]*" can grant access that capability. "*`logical`[p1050]*" 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 and document**:

1. Let *group* be a new browsing context group[p1050].
2. Append *group* to the user agent's browsing context group set[p1050].

3. Let *browsingContext* and *document* be the result of creating a new browsing context and document[p1047] with null, null, and *group*.
4. Append[p1051] *browsingContext* to *group*.
5. Return *group* and *document*.

To **append** a top-level browsing context[p1049] *browsingContext* to a browsing context group[p1050] *group*:

1. Append *browsingContext* to *group*'s browsing context set[p1050].
2. Set *browsingContext*'s group[p1050] to *group*.

To **remove** a top-level browsing context[p1049] *browsingContext*:

1. Assert: *browsingContext*'s group[p1050] is non-null.
2. Let *group* be *browsingContext*'s group[p1050].
3. Set *browsingContext*'s group[p1050] to null.
4. Remove *browsingContext* from *group*'s browsing context set[p1050].
5. If *group*'s browsing context set[p1050] is empty, then remove *group* from the user agent's browsing context group set[p1050].

Note

Append[p1051] and remove[p1051] are primitive operations that help define the lifetime of a browsing context group[p1050]. They are called by higher-level creation and destruction operations for Document[p135]s and browsing contexts[p1046].

When there are no Document[p135] objects whose browsing context[p1046] equals a given browsing context[p1046] (i.e., all such Document[p135]s have been destroyed[p1116]), and that browsing context[p1046]'s WindowProxy[p977] is eligible for garbage collection, then the browsing context[p1046] will never be accessed again. If it is a top-level browsing context[p1049], then at this point the user agent must remove[p1051] it.

7.3.3 Fully active documents §[p10 51]

A Document[p135] *d* is said to be **fully active** when *d* is the active document[p1036] of a navigable[p1035] *navigable*, and either *navigable* is a top-level traversable[p1037] or *navigable*'s container document[p1038] is fully active[p1051].

Because they are associated with an element, child navigables[p1039] are always tied to a specific Document[p135], their container document[p1038], in their parent navigable[p1035]. User agents must not allow the user to interact with child navigables[p1039] whose container documents[p1038] are not themselves fully active[p1051].

Example

The following example illustrates how a Document[p135] can be the active document[p1036] of its node navigable[p1036], while not being fully active[p1051]. Here `a.html` is loaded into a browser window, `b-1.html` starts out loaded into an `iframe`[p404] as shown, and `b-2.html` and `c.html` are omitted (they can simply be an empty document).

```
<!-- a.html -->
<!DOCTYPE html>
<html lang="en">
<title>Navigable A</title>

<iframe src="b-1.html"></iframe>
<button onclick="frames[0].location.href = 'b-2.html'">Click me</button>

<!-- b-1.html -->
<!DOCTYPE html>
<html lang="en">
<title>Navigable 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[p1036] of their respective node navigables[p1036]. They are also all fully active[p1051].

After clicking on the `button`[p585], and thus loading a new `Document`[p135] from `b-2.html` into navigable B, we have the following results:

- The `a.html` `Document`[p135] remains both the active document[p1036] of navigable A, and fully active[p1051].
- The `b-1.html` `Document`[p135] is now *not* the active document[p1036] of navigable B. As such it is also not fully active[p1051].
- The new `b-2.html` `Document`[p135] is now the active document[p1036] of navigable B, and is also fully active[p1051].
- The `c.html` `Document`[p135] is still the active document[p1036] of navigable C. However, since C's container document[p1038] is the `b-1.html` `Document`[p135], which is itself not fully active[p1051], this means the `c.html` `Document`[p135] is now not fully active[p1051].

7.4 Navigation and session history §[p1052]

Welcome to the dragon's maw. Navigation, session history, and the traversal through that session history are some of the most complex parts of this standard.

The basic concept may not seem so difficult:

- The user is looking at a navigable[p1035] that is presenting its active document[p1036]. They navigate[p1063] it to another URL.
- The browser fetches the given URL from the network, using it to populate[p1079] a new session history entry[p1053] with a newly-created[p1106] `Document`[p135].
- The browser updates the navigable[p1035]'s active session history entry[p1035] to the newly-populated one, and thus updates the active document[p1036] that it is showing to the user.
- At some point later, the user presses the browser back button[p1077] to go back to the previous session history entry[p1053].
- The browser looks at the URL[p1053] stored in that session history entry[p1053], and uses it to re-fetch and populate[p1079] that entry's document[p1053].
- The browser again updates the navigable[p1035]'s active session history entry[p1035].

You can see some of the intertwined complexity peeking through here, in how traversal can cause a navigation (i.e., a network fetch to a stored URL), and how a navigation necessarily needs to interface with the session history list to ensure that when it finishes the user is looking at the right thing. But the real problems come in with the various edge cases and interacting web platform features:

- Child navigables[p1039] (e.g., those contained in `iframe`[p404]s) can also navigate and traverse, but those navigations need to be linearized into a single session history list[p1037] since the user only has a single back/forward interface for the entire traversable navigable[p1037] (e.g., browser tab).
- Since the user can traverse back more than a single step in the session history (e.g., by holding down their back button), they can end up traversing multiple navigables[p1035] at the same time when child navigables[p1039] are involved. This needs to be synchronized across all of the involved navigables, which might involve multiple event loops[p1193] or even agent clusters.
- During navigation, servers can respond with 204 or 205 status codes or with `` `Content-Disposition: attachment` `` headers, which cause navigation to abort and the navigable[p1035] to stay on its original active document[p1046]. (This is much worse if it happens during a traversal-initiated navigation!)
- Various other HTTP headers, such as `` `Location` ``, `` `Refresh`[p1137] ``, `` `X-Frame-Options`[p1136] ``, and those for Content Security Policy, contribute to either the fetching process[p1082], or the `Document`-creation process[p1106], or both. The `` `Cross-Origin-Opener-Policy`[p946] `` header even contributes to the browsing context selection and creation[p947] process!
- Some navigations (namely fragment navigations[p1071] and single-page app navigations[p1078]) are synchronous, meaning that JavaScript code expects to observe the navigation's results instantly. This then needs to be synchronized with the view of the session history that all other navigables[p1035] in the tree see, which can be subject to race conditions and necessitate

resolving conflicting views of the session history.

- The platform has accumulated various exciting navigation-related features that need special-casing, such as `javascript:` URLs, `srcdoc` `iframe`s, and the `beforeunload` event.

In what follows, we have attempted to guide the reader through these complexities by appropriately cordoning them off into labeled sections and algorithms, and giving appropriate words of introduction where possible. Nevertheless, if you wish to truly understand navigation and session history, the usual advice will be invaluable.

7.4.1 Session history

7.4.1.1 Session history entries

A **session history entry** is a struct with the following items:

- **step**, a non-negative integer or "`pending`", initially "`pending`".
- **URL**, a URL
- **document state**, a document state.
- **classic history API state**, which is serialized state, initially StructuredSerializeForStorage(null).
- **navigation API state**, which is a serialized state, initially StructuredSerializeForStorage(undefined).
- **navigation API key**, which is a string, initially set to the result of generating a random UUID.
- **navigation API ID**, which is a string, initially set to the result of generating a random UUID.
- **scroll restoration mode**, a scroll restoration mode, initially "`auto`".
- **scroll position data**, which is scroll position data for the document's restorable scrollable regions.
- **persisted user state**, which is implementation-defined, initially null

Example

For example, some user agents might want to persist the values of form controls.

Note

User agents that persist the value of form controls are encouraged to also persist their directionality (the value of the element's `dir` attribute). This prevents values from being displayed incorrectly after a history traversal when the user had originally entered the values with an explicit, non-default directionality.

To get a session history entry's **document**, return its document state's document.

Serialized state is a serialization (via StructuredSerializeForStorage) 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) serialized state.

Pages can add serialized state to the session history. These are then deserialized and returned to the script when the user (or script) goes back in the history, thus enabling authors to use the "navigation" metaphor even in one-page applications.

Note

Serialized state 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` instance and it would have to be reconstructed if a new `Document` were opened.

An example of the latter would be something like keeping track of the precise coordinate from which a popup `div` was made to animate, so that if the user goes back, it can be made to animate to the same location. Or alternatively, it could be used to keep a pointer into a cache of data that would be fetched from the server based on the information in the URL, so that when going back

and forward, the information doesn't have to be fetched again.

A **scroll restoration mode** indicates whether the user agent should restore the persisted scroll position (if any) when traversing to an entry[p1053]. 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

7.4.1.2 Document state §[p10 54]

Document state holds state inside a session history entry[p1053] regarding how to present and, if necessary, recreate, a `Document`[p135]. It has:

- A **document**, a `Document`[p135] or null, initially null.

¶[p10 54] **Note**

When a history entry is active[p1035], it has a `Document`[p135] in its document state[p1053]. However, when a `Document`[p135] is not fully active[p1051], it's possible for it to be destroyed[p1116] to free resources. In such cases, this document[p1054] item will be nulled out. The URL[p1053] and other data in the session history entry[p1053] and document state[p1053] is then used to bring a new `Document`[p135] into being to take the place of the original, in the case where the user agent finds itself having to traverse to the entry.

If the `Document`[p135] is not *destroyed[p1116], then during history traversal[p1077], it can be reactivated[p1101]. The cache in which browsers store such `Document`[p135]s is often called a* back-forward cache*, or* bfcache *(or perhaps "blazingly fast" cache).*

- A **history policy container**, a policy container[p960] or null, initially null.
- A **request referrer**, which is "`no-referrer`", "`client`", or a URL, initially "`client`".
- A **request referrer policy**, which is a referrer policy, initially the default referrer policy.

  Note

  The request referrer policy[p1054] is distinct from the history policy container[p1054]'s referrer policy[p960]. The former is used for fetches of this document, whereas the latter controls fetches by this document.

- An **initiator origin**, which is an origin[p939] or null, initially null.
- An **origin**, which is an origin[p939] or null, initially null.

  Note

  This is the origin that we set "`about:`"-schemed `Document`[p135]s' origin to. We store it here because it is also used when restoring these `Document`[p135]s during traversal, since they are reconstructed locally without visiting the network. It is also used to compare the origin before and after the session history entry[p1053] is repopulated[p1079]. If the origins change, the navigable target name[p1055] is cleared.

- An **about base URL**, which is a URL or null, initially null.

  Note

  This will be populated only for "`about:`"-schemed `Document`[p135]s and will be the fallback base URL[p101] for those `Document`[p135]s. It is a snapshot of the initiator `Document`[p135]'s document base URL[p101].

- **Nested histories**, a list of nested histories[p1055], initially an empty list.
- A **resource**, a string, POST resource[p1055] or null, initially null.

Note

A string is treated as HTML. It's used to store the source of an `iframe srcdoc` document[p405].

- A **reload pending** boolean, initially false.
- An **ever populated** boolean, initially false.
- A **navigable target name** string, initially the empty string.
- A **not restored reasons**, a not restored reasons[p1035] or null, initially null.

User agents may destroy a document and its descendants[p1117] given the documents[p1054] of document states[p1054] with non-null documents[p1054], as long as the `Document`[p135] is not fully active[p1051].

Apart from that restriction, this standard does not specify when user agents should destroy the document[p1054] stored in a document state[p1054], versus keeping it cached.

A **POST resource** has:

- A **request body**, a byte sequence or failure.

  This is only ever accessed in parallel[p44], so it doesn't need to be stored in memory. However, it must return the same byte sequence each time. If this isn't possible due to resources changing on disk, or if resources can no longer be accessed, then this must be set to failure.

- A **request content-type**, which is `` `application/x-www-form-urlencoded` ``, `` `multipart/form-data`[p1577] ``, or `` `text/plain` ``.

A **nested history** has:

- An **id**, a unique internal value[p100].

  Note

  This is used to associate the nested history[p1055] with a navigable[p1035].

- **Entries**, a list of session history entries[p1053].

This will later contain ways to identify a child navigable across reloads.

Several contiguous entries in a session history can share the same document state[p1053]. This can occur when the initial entry is reached via normal navigation[p1063], and the following entry is added via `history.pushState()`[p989]. Or it can occur via navigation to a fragment[p1071].

Note

All entries that share the same document state[p1053] (and that are therefore merely different states of one particular document) are contiguous by construction.

A `Document`[p135] has a **latest entry**, a session history entry[p1053] or null.

Note

This is the entry that was most recently represented by a given `Document`[p135]. A single `Document`[p135] can represent many session history entries[p1053] over time, as many contiguous session history entries[p1053] can share the same document state[p1053] as explained above.

7.4.1.3 Centralized modifications of session history §p10 56

To maintain a single source of truth, all modifications to a traversable navigablep1037's session history entriesp1037 need to be synchronized. This is especially important due to how session history is influenced by all of the descendant navigablesp1035, and thus by multiple event loopsp1193. To accomplish this, we use the session history traversal parallel queuep1056 structure.

A **session history traversal parallel queue** is very similar to a parallel queuep44. It has an **algorithm set**, an ordered set.

The items in a session history traversal parallel queuep1056's algorithm setp1056 are either algorithm steps, or **synchronous navigation steps**, which are a particular brand of algorithm steps involving a **target navigable** (a navigablep1035).

To **append session history traversal steps** to a traversable navigablep1037 *traversable* given algorithm steps *steps*, append *steps* to *traversable*'s session history traversal queuep1037's algorithm setp1056.

To **append session history synchronous navigation steps** involving a navigablep1035 *targetNavigable* to a traversable navigablep1037 *traversable* given algorithm steps *steps*, append *steps* as synchronous navigation stepsp1056 targeting target navigablep1056 *targetNavigable* to *traversable*'s session history traversal queuep1037's algorithm setp1056.

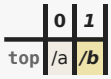
To **start a new session history traversal parallel queue**:

1. Let *sessionHistoryTraversalQueue* be a new session history traversal parallel queuep1056.
2. Run the following steps in parallelp44:
    1. While true:
        1. If *sessionHistoryTraversalQueue*'s algorithm setp1056 is empty, then continue.
        2. Let *steps* be the result of dequeuing from *sessionHistoryTraversalQueue*'s algorithm setp1056.
        3. Run *steps*.
3. Return *sessionHistoryTraversalQueue*.

Synchronous navigation stepsp1056 are tagged in the algorithm setp1056 to allow them to conditionally "jump the queue". This is handled within apply the history stepp1094.

¶p10 56 **Example**

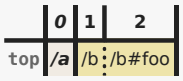
Imagine the joint session history depicted by this Jake diagramp1040:



And the following code runs at the top level:

```
history.back();
location.href = '#foo';
```

The desired result is:



This isn't straightforward, as the sync navigation wins the race in terms of being observable, whereas the traversal wins the race in terms of queuing steps on the session history traversal parallel queuep1056. To achieve this result, the following happens:

1. `history.back()`p989 appends stepsp1056 intended to traverse by a delta of −1.
2. `location.href = '#foo'`p982 synchronously changes the active session history entryp1035 entry to a newly-created one, with the URL `/b#foo`, and appends synchronous stepsp1056 to notify the central source of truth about that new entry. Note that this does *not* yet update the current session history entryp1035, current session history stepp1037, or the session history entriesp1037 list; those updates cannot be done synchronously, and instead must be done as part of the queued steps.

3. On the session history traversal parallel queue[p1056], the steps queued by `history.back()`[p989] run:

    1. The target history step is determined to be 0: the current session history step[p1037] (i.e., 1) plus the intended delta of −1.

    2. We enter the main apply the history step[p1091] algorithm.

       The entry at step 0, for the `/a` URL, has its document[p1053] populated[p1079].

       Meanwhile, the queue is checked for synchronous navigation steps[p1056]. The steps queued by the `location.href`[p982] setter now run, and block the traversal from performing effects beyond document population (such as, unloading documents and switching active history entries) until they are finished. Those steps cause the following to happen:

        1. The entry with URL `/b#foo` is added, with its step[p1053] determined to be 2: the current session history step[p1037] (i.e., 1) plus 1.

        2. We fully switch to that newly added entry, including a nested call to apply the history step[p1091]. This ultimately results in updating the document[p1100] by dispatching events like `hashchange`[p1575].

       Only once that is all complete, and the `/a` history entry has been fully populated with a document[p1053], do we move on with applying the history step given the target step of 0.

       At this point, the `Document`[p135] with URL `/b#foo` unloads[p1114], and we finish moving to our target history step 0, which makes the entry with URL `/a` become the active session history entry[p1035] and 0 become the current session history step[p1037].

¶[p1057] **Example**

Here is another more complex example, involving races between populating two different `iframe`[p404]s, and a synchronous navigation once one of those iframes loads. We start with this setup:

	0	1	2
top	***/t***		
frames[0]	/i-0-a	***/i-0-b***	
frames[1]	/i-1-a		***/i-1-b***

and then call `history.go(-2)`[p989]. The following then occurs:

1. `history.go(-2)`[p989] appends steps[p1056] intended to traverse by a delta of −2. Once those steps run:

    1. The target step is determined to be 2 + (−2) = 0.

    2. In parallel, the fetches are made to populate[p1079] the two iframes, fetching `/i-0-a` and `/i-1-a` respectively.

       Meanwhile, the queue is checked for synchronous navigation steps[p1056]. There aren't any right now.

    3. In the fetch race, the fetch for `/i-0-a` wins. We proceed onward to finish all of apply the history step[p1091]'s work for how the traversal impacts the `frames[0]` navigable[p1035], including updating its active session history entry[p1035] to the entry with URL `/i-0-a`.

    4. Before the fetch for `/i-1-a` finishes, we reach the point where scripts may run for the newly-created document[p1101] in the `frames[0]` navigable[p1035]'s active document[p1036]. Some such script does run:

       ```
       location.href = '#foo'
       ```

       This synchronously changes the `frames[0]` navigable's active session history entry[p1035] entry to a newly-created one, with the URL `/i-0-a#foo`, and appends synchronous steps[p1056] to notify the central source of truth about that new entry.

       Unlike in the previous example[p1056], these synchronous steps do *not* "jump the queue" and update the traversable[p1037] before we finish the fetch for `/i-1-a`. This is because the navigable in question, `frames[0]`, has already been altered as part of the traversal, so we know that with the current session history step[p1037]

being 2, adding the new entry as a step 3 doesn't make sense.

5. Once the fetch for `/i-1-a` finally finishes, we proceed to finish updating the `frames[1]` navigable[p1035] for the traversal, including updating its active session history entry[p1035] to the entry with URL `/i-1-a`.

6. Now that both navigables have finished processing the traversal, we update the current session history step[p1037] to the target step of 0.

2. Now we can process the steps that were queued for the synchronous navigation:

    1. The `/i-0-a#foo` entry is added, with its step[p1053] determined to be 1: the current session history step[p1037] (i.e., 0) plus 1. This also clears existing forward history[p1059].

    2. We fully switch to that newly added entry, including calling apply the history step[p1091]. This ultimately results in updating the document[p1100] by dispatching events like `hashchange`[p1575], as well as updating the current session history step[p1037] to the target step of 1.

The end result is:

	0	1
top	/t	
frames[0]	/i-0-a	/i-0-a#foo
frames[1]	/i-1-a	

7.4.1.4 Low-level operations on session history §[p1058]

This section contains a miscellaneous grab-bag of operations that we perform throughout the standard when manipulating session history. The best way to get a sense of what they do is to look at their call sites.

To **get session history entries** of a navigable[p1035] *navigable*:

1. Let *traversable* be *navigable*'s traversable navigable[p1037].
2. Assert: this is running within *traversable*'s session history traversal queue[p1037].
3. If *navigable* is *traversable*, return *traversable*'s session history entries[p1037].
4. Let *docStates* be an empty ordered set of document states[p1054].
5. For each *entry* of *traversable*'s session history entries[p1037], append *entry*'s document state[p1053] to *docStates*.
6. For each *docState* of *docStates*:
    1. For each *nestedHistory* of *docState*'s nested histories[p1054]:
        1. If *nestedHistory*'s id[p1055] equals *navigable*'s id[p1035], return *nestedHistory*'s entries[p1055].
        2. For each *entry* of *nestedHistory*'s entries[p1055], append *entry*'s document state[p1053] to *docStates*.
7. Assert: this step is not reached.

To **get session history entries for the navigation API** of a navigable[p1035] *navigable* given an integer *targetStep*:

1. Let *rawEntries* be the result of getting session history entries[p1058] for *navigable*.
2. Let *entriesForNavigationAPI* be a new empty list.
3. Let *startingIndex* be the index of the session history entry[p1053] in *rawEntries* who has the greatest step[p1053] less than or equal to *targetStep*.

    Note

    See this example[p1099] to understand why it's the greatest step less than or equal to targetStep.

4. Append *rawEntries*[*startingIndex*] to *entriesForNavigationAPI*.

5. Let *startingOrigin* be *rawEntries*[*startingIndex*]'s document state[p1053]'s origin[p1054].
6. Let *i* be *startingIndex* − 1.
7. While *i* > 0:
    1. If *rawEntries*[*i*]'s document state[p1053]'s origin[p1054] is not same origin[p940] with *startingOrigin*, then break.
    2. Prepend *rawEntries*[*i*] to *entriesForNavigationAPI*.
    3. Set *i* to *i* − 1.
8. Set *i* to *startingIndex* + 1.
9. While *i* < *rawEntries*'s size:
    1. If *rawEntries*[*i*]'s document state[p1053]'s origin[p1054] is not same origin[p940] with *startingOrigin*, then break.
    2. Append *rawEntries*[*i*] to *entriesForNavigationAPI*.
    3. Set *i* to *i* + 1.
10. Return *entriesForNavigationAPI*.

To **clear the forward session history** of a traversable navigable[p1037] *navigable*:

1. Assert: this is running within *navigable*'s session history traversal queue[p1037].
2. Let *step* be the *navigable*'s current session history step[p1037].
3. Let *entryLists* be the ordered set « *navigable*'s session history entries[p1037] ».
4. For each *entryList* of *entryLists*:
    1. Remove every session history entry[p1053] from *entryList* that has a step[p1053] greater than *step*.
    2. For each *entry* of *entryList*:
        1. For each *nestedHistory* of *entry*'s document state[p1053]'s nested histories[p1054], append *nestedHistory*'s entries list[p1055] to *entryLists*.

To **get all used history steps** that are part of traversable navigable[p1037] *traversable*:

1. Assert: this is running within *traversable*'s session history traversal queue[p1037].
2. Let *steps* be an empty ordered set of non-negative integers.
3. Let *entryLists* be the ordered set « *traversable*'s session history entries[p1037] ».
4. For each *entryList* of *entryLists*:
    1. For each *entry* of *entryList*:
        1. Append *entry*'s step[p1053] to *steps*.
        2. For each *nestedHistory* of *entry*'s document state[p1053]'s nested histories[p1054], append *nestedHistory*'s entries list[p1055] to *entryLists*.
5. Return *steps*, sorted.

7.4.2 Navigation §[p1059]

Certain actions cause a navigable[p1035] to *navigate*[p1063] to a new resource.

Example

For example, following a hyperlink[p321], form submission[p658], and the `window.open()`[p969] and `location.assign()`[p985] methods can all cause navigation.

¶ p10 60 Note

Although in this standard the word "navigation" refers specifically to the navigate p1063 algorithm, this doesn't always line up with web developer or user perceptions. For example:

- *The URL and history update steps p1078 are often used during so-called "single-page app navigations" or "same-document navigations", but they do not trigger the navigate p1063 algorithm.*
- *Reloads p1077 and traversals p1077 are sometimes talked about as a type of navigation, since all three will often attempt to populate the history entry's document p1079 and thus could perform navigational fetches. See, e.g., the APIs exposed* Navigation Timing*. But they have their own entry point algorithms, separate from the navigate p1063 algorithm. [NAVIGATIONTIMING] p1584*
- *Although fragment navigations p1071 are always done through the navigate p1063 algorithm, a user might perceive them as more like jumping around a single page, than as a true navigation.*

7.4.2.1 Supporting concepts § p10 60

Before we can jump into the navigation algorithm p1063 itself, we need to establish several important structures that it uses.

The **source snapshot params** struct is used to capture data from a Document p135 initiating a navigation. It is snapshotted at the beginning of a navigation and used throughout the navigation's lifetime. It has the following items:

has transient activation
: a boolean

sandboxing flags
: a sandboxing flag set p957

allows downloading
: a boolean

fetch client
: an environment settings object p1144 or null, only to be used as a request client

source policy container
: a policy container p960

To **snapshot source snapshot params** given a Document p135-or-null *sourceDocument*:

1. If *sourceDocument* is null, then return a new source snapshot params p1060 with

   has transient activation p1060
   : true

   sandboxing flags p1060
   : an empty sandboxing flag set p957

   allows downloading p1060
   : true

   fetch client p1060
   : null

   source policy container p1060
   : a new policy container p960

   Note

   This only occurs p1063 in the case of a browser UI-initiated navigation.

2. Return a new source snapshot params p1060 with

   has transient activation p1060
   : true if *sourceDocument*'s relevant global object p1151 has transient activation p868; otherwise false

   sandboxing flags p1060
   : *sourceDocument*'s active sandboxing flag set p959

   allows downloading p1060
   : false if *sourceDocument*'s active sandboxing flag set p959 has the sandboxed downloads browsing context flag p958 set;

otherwise true

fetch client[p1060]

sourceDocument's relevant settings object[p1151]

source policy container[p1060]

a clone[p960] of *sourceDocument*'s policy container[p136]

The **target snapshot params** struct is used to capture data from a navigable[p1035] being navigated. Like source snapshot params[p1060], it is snapshotted at the beginning of a navigation and used throughout the navigation's lifetime. It has the following items:

sandboxing flags

a sandboxing flag set[p957]

`iframe` ***element referrer policy***

a referrer policy

To **snapshot target snapshot params** given a navigable[p1035] *targetNavigable*, return a new target snapshot params[p1061] with:

sandboxing flags[p1061]

the result of determining the creation sandboxing flags[p960] given *targetNavigable*'s active browsing context[p1036] and *targetNavigable*'s container[p1038]

`iframe` element referrer policy[p1061]

the result of determining the `iframe` element referrer policy[p960] given *targetNavigable*'s container[p1038]

Much of the navigation process is concerned with determining how to create a new `Document`[p135], which ultimately happens in the create and initialize a `Document` object[p1106] algorithm. The parameters to that algorithm are tracked via a **navigation params** struct, which has the following items:

id

null or a navigation ID[p1062]

navigable

the navigable[p1035] to be navigated

request

null or a request that started the navigation

response

a response that ultimately was navigated to (potentially a network error)

fetch controller

null or a fetch controller

commit early hints

null or an algorithm accepting a `Document`[p135], once it has been created

COOP enforcement result

an opener policy enforcement result[p948], used for reporting and potentially for causing a browsing context group switch[p947]

reserved environment

null or an environment[p1143] reserved for the new `Document`[p135]

origin

an origin[p939] to use for the new `Document`[p135]

policy container

a policy container[p960] to use for the new `Document`[p135]

final sandboxing flag set

a sandboxing flag set[p957] to impose on the new `Document`[p135]

`iframe` ***element referrer policy***

a referrer policy, used in the internal ancestor origin objects list creation steps[p137]

opener policy

an opener policy[p945] to use for the new `Document`[p135]

navigation timing type

a NavigationTimingType used for creating the navigation timing entry for the new Document[p135]

about base URL

a URL or null used to populate the new Document[p135]'s about base URL[p136]

user involvement

a user navigation involvement[p1062] used when obtaining a browsing context[p949] for the new Document[p135]

Note

Once a navigation params[p1061] struct is created, this standard does not mutate any of its items. They are only passed onward to other algorithms.

A **navigation ID** is a UUID string generated during navigation. It is used to interface with the *WebDriver BiDi* specification as well as to track the ongoing navigation[p1076]. [WEBDRIVERBIDI][p1588]

After Document[p135] creation, the relevant traversable navigable[p1037]'s session history[p1037] gets updated. The NavigationHistoryBehavior[p995] enumeration is used to indicate the desired type of session history update to the navigate[p1063] algorithm. It is one of the following:

"push"

A regular navigation which adds a new session history entry[p1053], and will clear the forward session history[p1059].

"replace"

A navigation that will replace the active session history entry[p1035].

"auto"

The default value, which will be converted very early in the navigate[p1063] algorithm into "push[p1062]" or "replace[p1062]". Usually it becomes "push[p1062]", but under certain circumstances[p1064] it becomes "replace[p1062]" instead.

A **history handling behavior** is a NavigationHistoryBehavior[p995] that is either "push[p1062]" or "replace[p1062]", i.e., that has been resolved away from any initial "auto[p1062]" value.

The navigation must be a replace, given a URL *url* and a Document[p135] *document*, if any of the following are true:

- *url*'s scheme is "javascript[p1068]"; or
- *document*'s is initial about:blank[p136] is true.

Note

Other cases that often, but not always, force a "replace[p1062]" navigation are:

- *if the Document[p135] is not completely loaded[p1114]; or*
- *if the target URL equals the Document[p135]'s URL.*

Various parts of the platform track whether a user is involved in a navigation. A **user navigation involvement** is one of the following:

"browser UI"

The navigation was initiated by the user via browser UI mechanisms.

"activation"

The navigation was initiated by the user via the activation behavior of an element.

"none"

The navigation was not initiated by the user.

For convenience at certain call sites, the **user navigation involvement** for an Event *event* is defined as follows:

1. Assert: this algorithm is being called as part of an activation behavior definition.

2. Assert: *event*'s `type` is "`click`".
3. If *event*'s `isTrusted` is initialized to true, then return "`activation`".
4. Return "`none`".

7.4.2.2 Beginning navigation

To **navigate** a navigable *navigable* to a URL *url* using an optional `Document`-or-null *sourceDocument* (default null), with an optional POST resource, string, or null ***documentResource*** (default null), an optional response-or-null ***response*** (default null), an optional boolean ***exceptionsEnabled*** (default false), an optional `NavigationHistoryBehavior` ***historyHandling*** (default "`auto`"), an optional serialized state-or-null ***navigationAPIState*** (default null), an optional entry list or null ***formDataEntryList*** (default null), an optional referrer policy ***referrerPolicy*** (default the empty string), an optional user navigation involvement ***userInvolvement*** (default "`none`"), an optional `Element` ***sourceElement*** (default null), an optional boolean ***initialInsertion*** (default false), and an optional navigation API method tracker-or-null ***apiMethodTracker*** (default null):

1. Let *cspNavigationType* be "`form-submission`" if *formDataEntryList* is non-null; otherwise "`other`".
2. Let *sourceSnapshotParams* be the result of snapshotting source snapshot params given *sourceDocument*.
3. Let *initiatorOriginSnapshot* be a new opaque origin.
4. Let *initiatorBaseURLSnapshot* be `about:blank`.
5. If *sourceDocument* is null:
    1. Assert: *userInvolvement* is "`browser UI`".
    2. If *url*'s scheme is "`javascript`", then set *initiatorOriginSnapshot* to *navigable*'s active document's origin.
6. Otherwise:
    1. Assert: *userInvolvement* is not "`browser UI`".
    2. If *sourceDocument*'s node navigable is not allowed by sandboxing to navigate *navigable* given *sourceSnapshotParams*:
        1. If *exceptionsEnabled* is true, then throw a "`SecurityError`" `DOMException`.
        2. Return.
    3. Set *initiatorOriginSnapshot* to *sourceDocument*'s origin.
    4. Set *initiatorBaseURLSnapshot* to *sourceDocument*'s document base URL.
7. Let *navigationId* be the result of generating a random UUID. [WEBCRYPTO]
8. If the surrounding agent is equal to *navigable*'s active document's relevant agent, then continue these steps. Otherwise, queue a global task on the navigation and traversal task source given *navigable*'s active window to continue these steps.

    Note

    We do this because we are about to look at a lot of properties of navigable's *active document, which are in theory only accessible over in the appropriate event loop. (But, we do not want to unconditionally queue a task, since — for example — same-event-loop fragment navigations need to take effect synchronously.)*

    Another implementation strategy would be to replicate the relevant information across event loops, or into a canonical "browser process", so that it can be consulted without queueing a task. This could give different results than what we specify here in edge cases, where the relevant properties have changed over in the target event loop but not yet been replicated. Further testing is needed to determine which of these strategies best matches browser behavior, in such racy edge cases.

9. If *navigable*'s active document's unload counter is greater than 0, then invoke WebDriver BiDi navigation failed with *navigable* and a WebDriver BiDi navigation status whose id is *navigationId*, status is "`canceled`", and url is *url*, and return.
10. Let *container* be *navigable*'s container.

11. If *container* is an `iframe`[p404] element and will lazy load element steps[p105] given *container* returns true, then stop intersection-observing a lazy loading element[p107] *container* and set *container*'s lazy load resumption steps[p105] to null.

12. If *navigable*'s allowed to perform a navigation or history update[p1036] returns blocked, then invoke WebDriver BiDi navigation failed with *navigable* and a WebDriver BiDi navigation status whose id is *navigationId*, status is "`canceled`", and url is *url*, and return.

13. If *historyHandling* is "`auto`[p1062]":

    1. If *url* equals *navigable*'s active document[p1036]'s URL, and either *userInvolvement* is "`browser UI`[p1062]" or *initiatorOriginSnapshot* is same origin[p940] with *navigable*'s active document[p1036]'s origin, then set *historyHandling* to "`replace`[p1062]".

    2. Otherwise, set *historyHandling* to "`push`[p1062]".

14. If the navigation must be a replace[p1062] given *url* and *navigable*'s active document[p1036], then set *historyHandling* to "`replace`[p1062]".

15. If all of the following are true:

    - *documentResource* is null;
    - *response* is null;
    - *url* equals *navigable*'s active session history entry[p1035]'s URL[p1053] with *exclude fragments* set to true; and
    - *url*'s fragment is non-null,

    then:

    1. Navigate to a fragment[p1071] given *navigable*, *url*, *historyHandling*, *userInvolvement*, *sourceElement*, *navigationAPIState*, and *navigationId*.

    2. Return.

16. If *navigable*'s parent[p1035] is non-null, then set *navigable*'s is delaying load events[p1036] to true.

17. Let *targetSnapshotParams* be the result of snapshotting target snapshot params[p1061] given *navigable*.

18. Invoke WebDriver BiDi navigation started with *navigable* and a new WebDriver BiDi navigation status whose id is *navigationId*, status is "`pending`", and url is *url*.

19. If *navigable*'s ongoing navigation[p1076] is "`traversal`":

    1. Invoke WebDriver BiDi navigation failed with *navigable* and a new WebDriver BiDi navigation status whose id is *navigationId*, status is "`canceled`", and url is *url*.

    2. Return.

    Note

    Any attempts to navigate a navigable[p1035] that is currently traversing[p1090] are ignored.

20. Set the ongoing navigation[p1077] for *navigable* to *navigationId*.

    Note

    This will have the effect of aborting other ongoing navigations of navigable*, since at certain points during navigation changes to the ongoing navigation[p1076] will cause further work to be abandoned.*

21. If *url*'s scheme is "`javascript`[p1068]":

    1. Let *request* be a new request, with

       URL
       url
       client
       sourceSnapshotParams's fetch client[p1060]
       policy container
       sourceSnapshotParams's source policy container[p1060]

Note

This is a synthetic request, needed because the Content Security Policy check in navigate to a `javascript:` URL operates on a request. It will never hit the network.

2. Queue a global task on the navigation and traversal task source given *navigable*'s active window to navigate to a `javascript:` URL given *navigable*, *request*, *historyHandling*, *initiatorOriginSnapshot*, *userInvolvement*, *cspNavigationType*, *initialInsertion*, and *navigationId*.

3. Return.

22. If all of the following are true:

    - *userInvolvement* is not "`browser UI`";
    - *navigable*'s active document's origin is same origin-domain with *sourceDocument*'s origin;
    - *navigable*'s active document's is initial `about:blank` is false; and
    - *url*'s scheme is a fetch scheme,

    then:

    1. Let *navigation* be *navigable*'s active window's navigation API.
    2. Let *entryListForFiring* be *formDataEntryList* if *documentResource* is a POST resource; otherwise, null.
    3. Let *navigationAPIStateForFiring* be *navigationAPIState* if *navigationAPIState* is not null; otherwise, StructuredSerializeForStorage(undefined).
    4. Let *continue* be the result of firing a push/replace/reload `navigate` event at *navigation* with *navigationType* set to *historyHandling*, *isSameDocument* set to false, *userInvolvement* set to *userInvolvement*, *sourceElement* set to *sourceElement*, *formDataEntryList* set to *entryListForFiring*, *destinationURL* set to *url*, *navigationAPIState* set to *navigationAPIStateForFiring*, and *apiMethodTracker* set to *apiMethodTracker*.
    5. If *continue* is false, then return.

    Note

    It is possible for navigations with userInvolvement *of* "`browser UI`" *or initiated by a cross origin-domain* sourceDocument *to fire* `navigate` *events, if they go through the earlier navigate to a fragment path.*

23. If *sourceDocument* is *navigable*'s container document, then reserve deferred fetch quota for *navigable*'s container given *url*'s origin.

24. In parallel, run these steps:

    1. Let *unloadPromptCanceled* be the result of checking if unloading is canceled for *navigable*'s active document's inclusive descendant navigables.
    2. If *unloadPromptCanceled* is not "`continue`", or *navigable*'s ongoing navigation is no longer *navigationId*:
        1. Invoke WebDriver BiDi navigation failed with *navigable* and a new WebDriver BiDi navigation status whose id is *navigationId*, status is "`canceled`", and url is *url*.
        2. Abort these steps.
    3. Queue a global task on the navigation and traversal task source given *navigable*'s active window to abort a document and its descendants given *navigable*'s active document.
    4. Let *documentState* be a new document state with

    request referrer policy
    referrerPolicy
    initiator origin
    initiatorOriginSnapshot
    resource
    documentResource
    navigable target name
    navigable's target name

Note

The navigable target name[p1055] can get cleared under various conditions later in the navigation process, before the document state is finalized.

5. If *url* matches about:blank[p100] or is about:srcdoc[p100]:

    1. Set *documentState*'s origin[p1054] to *initiatorOriginSnapshot*.

    2. Set *documentState*'s about base URL[p1054] to *initiatorBaseURLSnapshot*.

6. Let *historyEntry* be a new session history entry[p1053], with its URL[p1053] set to *url* and its document state[p1053] set to *documentState*.

7. Let *navigationParams* be null.

8. If *response* is non-null:

¶[p1066] Note

The navigate[p1063] algorithm is only supplied with a response as part of the object[p416] and embed[p413] processing models, or for processing parts of multipart/x-mixed-replace responses[p1111] after the initial response.

    1. Let *sourcePolicyContainer* be a clone[p960] of the *sourceDocument*'s policy container[p136], if *sourceDocument* is not null; otherwise null.

    2. Let *policyContainer* be the result of determining navigation params policy container[p961] given *response*'s URL, null, *sourcePolicyContainer*, *navigable*'s container document[p1038]'s policy container[p136], and null.

    3. Let *finalSandboxFlags* be the union of *targetSnapshotParams*'s sandboxing flags[p1061] and *policyContainer*'s CSP list[p960]'s CSP-derived sandboxing flags[p959].

    4. Let *responseOrigin* be the result of determining the origin[p1049] given *response*'s URL, *finalSandboxFlags*, and *documentState*'s initiator origin[p1054].

    5. Let *coop* be a new opener policy[p945].

    6. Let *coopEnforcementResult* be a new opener policy enforcement result[p948] with

        url[p948]
        : *response*'s URL

        origin[p948]
        : *responseOrigin*

        opener policy[p948]
        : *coop*

    7. Set *navigationParams* to a new navigation params[p1061], with

        id[p1061]
        : *navigationId*

        navigable[p1061]
        : *navigable*

        request[p1061]
        : null

        response[p1061]
        : *response*

        fetch controller[p1061]
        : null

        commit early hints[p1061]
        : null

        COOP enforcement result[p1061]
        : *coopEnforcementResult*

        reserved environment[p1061]
        : null

        origin[p1061]
        : *responseOrigin*

        policy container[p1061]
        : *policyContainer*

final sandboxing flag set[p1061]
finalSandboxFlags
`iframe` element referrer policy[p1061]
targetSnapshotParams's `iframe` element referrer policy[p1061]
opener policy[p1061]
coop
navigation timing type[p1062]
"`navigate`"
about base URL[p1062]
documentState's about base URL[p1054]
user involvement[p1062]
userInvolvement

9. Attempt to populate the history entry's document[p1079] for *historyEntry*, given *navigable*, "`navigate`", *sourceSnapshotParams*, *targetSnapshotParams*, *userInvolvement*, *navigationId*, *navigationParams*, *cspNavigationType*, with *allowPOST*[p1079] set to true and *completionSteps*[p1079] set to the following step:

    1. Append session history traversal steps[p1056] to *navigable*'s traversable navigable[p1037] to finalize a cross-document navigation[p1067] given *navigable*, *historyHandling*, *userInvolvement*, and *historyEntry*.

7.4.2.3 Ending navigation §[p10 67]

Although the usual cross-document navigation case will first foray into populating a session history entry[p1079] with a `Document`[p135], all navigations that don't get aborted will ultimately end up calling into one of the below algorithms.

7.4.2.3.1 The usual cross-document navigation case §[p10 67]

To **finalize a cross-document navigation** given a navigable[p1035] *navigable*, a history handling behavior[p1062] *historyHandling*, a user navigation involvement[p1062] *userInvolvement*, and a session history entry[p1053] *historyEntry*:

1. Assert: this is running on *navigable*'s traversable navigable[p1037]'s session history traversal queue[p1037].
2. Set *navigable*'s is delaying `load` events[p1036] to false.
3. If *historyEntry*'s document[p1053] is null, then return.

    Note

    This means that attempting to populate the history entry's document[p1079] ended up not creating a document, as a result of e.g., the navigation being canceled by a subsequent navigation, a 204 No Content response, etc.

4. If all of the following are true:

    - *navigable*'s parent[p1035] is null;
    - *historyEntry*'s document[p1053]'s browsing context[p1046] is not an auxiliary browsing context[p1046] whose opener browsing context[p1046] is non-null; and
    - *historyEntry*'s document[p1053]'s origin is not same origin[p940] with *navigable*'s active document[p1036]'s origin,

    then set *historyEntry*'s document state[p1053]'s navigable target name[p1055] to the empty string.
5. Let *entryToReplace* be *navigable*'s active session history entry[p1035] if *historyHandling* is "`replace`[p1062]", otherwise null.
6. Let *traversable* be *navigable*'s traversable navigable[p1037].
7. Let *targetStep* be null.
8. Let *targetEntries* be the result of getting session history entries[p1058] for *navigable*.
9. If *entryToReplace* is null:

    1. Clear the forward session history[p1059] of *traversable*.

2. Set *targetStep* to *traversable*'s current session history step[p1037] + 1.
3. Set *historyEntry*'s step[p1053] to *targetStep*.
4. Append *historyEntry* to *targetEntries*.

Otherwise:

1. Replace *entryToReplace* with *historyEntry* in *targetEntries*.
2. Set *historyEntry*'s step[p1053] to *entryToReplace*'s step[p1053].
3. If *historyEntry*'s document state[p1053]'s origin[p1054] is same origin[p940] with *entryToReplace*'s document state[p1053]'s origin[p1054], then set *historyEntry*'s navigation API key[p1053] to *entryToReplace*'s navigation API key[p1053].
4. Set *targetStep* to *traversable*'s current session history step[p1037].

10. Apply the push/replace history step[p1090] *targetStep* to *traversable* given *historyHandling* and *userInvolvement*.

7.4.2.3.2 The `javascript:` URL special case §[p10 68]

`javascript:`[p1068] URLs have a dedicated label on the issue tracker documenting various problems with their specification.

To **navigate to a `javascript:` URL**, given a navigable[p1035] *targetNavigable*, a request *request*, a history handling behavior[p1062] *historyHandling*, an origin[p939] *initiatorOrigin*, a user navigation involvement[p1062] *userInvolvement*, a string *cspNavigationType*, a boolean *initialInsertion*, and a navigation ID[p1062] *navigationId*:

1. Assert: *historyHandling* is "`replace`[p1062]".
2. If *targetNavigable*'s ongoing navigation[p1076] is no longer *navigationId*, then return.
3. Set the ongoing navigation[p1077] for *targetNavigable* to null.
4. If *initiatorOrigin* is not same origin-domain[p940] with *targetNavigable*'s active document[p1036]'s origin, then return.
5. If the result of should navigation request of type be blocked by Content Security Policy? given *request* and *cspNavigationType* is "`Blocked`", then return. [CSP][p1580]

   Note

   This can change request's *URL, as the `require-trusted-types-for` pre-navigation check passes the script source through the default policy. [TRUSTED-TYPES][p1587]*
6. Let *newDocument* be the result of evaluating a `javascript:` URL[p1069] given *targetNavigable*, *request*'s URL, *initiatorOrigin*, *userInvolvement*, and *navigationId*.
7. If *newDocument* is null:
   1. Let *container* be *targetNavigable*'s container[p1038].
   2. If *initialInsertion* is true, *container* is non-null, and *targetNavigable*'s active document[p1036]'s is initial `about:blank`[p136] is true:
      1. If *container* is an `iframe`[p404] element, then run the iframe load event steps[p408] given *container*.
      2. Otherwise, fire an event named `load`[p1575] at *container*.

      Note

      container *can be null if author code, such as the Trusted Types default policy or* `javascript:frameElement.remove()`*, removed it.*
   3. Return.

Note

In this case, some JavaScript code was executed, but no new `Document`[p135] *was created, so we will not perform a*

navigation.

8. Assert: *initiatorOrigin* is *newDocument*'s origin.

9. Let *entryToReplace* be *targetNavigable*'s active session history entry[p1035].

10. Let *oldDocState* be *entryToReplace*'s document state[p1053].

11. Let *documentState* be a new document state[p1054] with

    document[p1054]
    newDocument
    history policy container[p1054]
    a clone[p960] of the *oldDocState*'s history policy container[p1054] if it is non-null; null otherwise
    request referrer[p1054]
    oldDocState's request referrer[p1054]
    request referrer policy[p1054]
    oldDocState's request referrer policy[p1054] or should this be the *referrerPolicy* that was passed to navigate[p1063]?
    initiator origin[p1054]
    initiatorOrigin
    origin[p1054]
    initiatorOrigin
    about base URL[p1054]
    oldDocState's about base URL[p1054]
    resource[p1054]
    null
    ever populated[p1055]
    true
    navigable target name[p1055]
    oldDocState's navigable target name[p1055]

12. Let *historyEntry* be a new session history entry[p1053], with

    URL[p1053]
    entryToReplace's URL[p1053]
    document state[p1053]
    documentState

    Note

    For the URL[p1053], we do not *use* request's *URL, i.e., the* `javascript:`[p1068] *URL that was evaluated. This means* `javascript:`[p1068] *URLs are never stored in session history, and so can never be traversed to.*

13. Append session history traversal steps[p1056] to *targetNavigable*'s traversable navigable[p1037] to finalize a cross-document navigation[p1067] with *targetNavigable*, *historyHandling*, *userInvolvement*, and *historyEntry*.

To **evaluate a `javascript:` URL** given a navigable[p1035] *targetNavigable*, a URL *url*, an origin[p939] *newDocumentOrigin*, a user navigation involvement[p1062] *userInvolvement*, and a navigation ID[p1062] *navigationId*:

1. Let *urlString* be the result of running the URL serializer on *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. Let *settings* be *targetNavigable*'s active document[p1036]'s relevant settings object[p1151].

5. Let *baseURL* be *settings*'s API base URL[p1144].

6. Let *script* be the result of creating a classic script[p1161] given *scriptSource*, *settings*, *baseURL*, and the default script fetch options[p1154].

7. Let *evaluationStatus* be the result of running the classic script[p1165] *script*.

8. Let *result* be null.

9. If *evaluationStatus* is a normal completion, and *evaluationStatus*.[[Value]] is a String, then set *result* to

evaluationStatus.[[Value]].

10. Otherwise, return null.

11. Let *response* be a new response with

    URL
    targetNavigable's active document[p1036]'s URL
    header list
    « (`Content-Type[p102]`, `text/html;charset=utf-8`) »
    body
    the UTF-8 encoding of *result*, as a body

    Note

    The encoding to UTF-8 means that unpaired surrogates will not roundtrip, once the HTML parser decodes the response body.

12. Let *policyContainer* be *targetNavigable*'s active document[p1036]'s policy container[p136].

13. Let *finalSandboxFlags* be *policyContainer*'s CSP list[p960]'s CSP-derived sandboxing flags[p959].

14. Let *coop* be *targetNavigable*'s active document[p1036]'s opener policy[p136].

15. Let *coopEnforcementResult* be a new opener policy enforcement result[p948] with

    url[p948]
    url
    origin[p948]
    newDocumentOrigin
    opener policy[p948]
    coop

16. Let *navigationParams* be a new navigation params[p1061], with

    id[p1061]
    navigationId
    navigable[p1061]
    targetNavigable
    request[p1061]
    null this will cause the referrer of the resulting Document[p135] to be null; is that correct?
    response[p1061]
    response
    fetch controller[p1061]
    null
    commit early hints[p1061]
    null
    COOP enforcement result[p1061]
    coopEnforcementResult
    reserved environment[p1061]
    null
    origin[p1061]
    newDocumentOrigin
    policy container[p1061]
    policyContainer
    final sandboxing flag set[p1061]
    finalSandboxFlags
    iframe element referrer policy[p1061]
    targetSnapshotParams's iframe element referrer policy[p1061]
    opener policy[p1061]
    coop
    navigation timing type[p1062]
    "navigate"
    about base URL[p1062]
    targetNavigable's active document[p1036]'s about base URL[p136]
    user involvement[p1062]
    userInvolvement

17. Return the result of loading an HTML document[p1110] given *navigationParams*.

7.4.2.3.3 Fragment navigations §[p1071]

To **navigate to a fragment** given a navigable[p1035] *navigable*, a URL *url*, a history handling behavior[p1062] *historyHandling*, a user navigation involvement[p1062] *userInvolvement*, an `Element`-or-null *sourceElement*, a serialized state[p1053]-or-null *navigationAPIState*, and a navigation ID[p1062] *navigationId*:

1. Let *navigation* be *navigable*'s active window[p1036]'s navigation API[p995].
2. Let *destinationNavigationAPIState* be *navigable*'s active session history entry[p1035]'s navigation API state[p1053].
3. If *navigationAPIState* is not null, then set *destinationNavigationAPIState* to *navigationAPIState*.
4. Let *continue* be the result of firing a push/replace/reload `navigate` event[p1019] at *navigation* with *navigationType*[p1019] set to *historyHandling*, *isSameDocument*[p1019] set to true, *userInvolvement*[p1019] set to *userInvolvement*, *sourceElement*[p1019] set to *sourceElement*, *destinationURL*[p1019] set to *url*, and *navigationAPIState*[p1019] set to *destinationNavigationAPIState*.
5. If *continue* is false:
    1. If *navigation*'s ongoing `navigate` event[p1007] is not null, then set the ongoing navigation[p1077] of *navigable* to *navigationId*.

        Note

        This makes intercepted hash navigations cancelable by browser UI or `window.stop()`[p971].

    2. Return.
6. Let *historyEntry* be a new session history entry[p1053], with

    URL[p1053]
    : *url*

    document state[p1053]
    : *navigable*'s active session history entry[p1035]'s document state[p1053]

    navigation API state[p1053]
    : *destinationNavigationAPIState*

    scroll restoration mode[p1053]
    : *navigable*'s active session history entry[p1035]'s scroll restoration mode[p1053]

    Note

    For navigations performed with `navigation.navigate()`[p1004], the value provided by the `state`[p995] option is used for the new navigation API state[p1053]. (This will set it to the serialization of undefined, if no value is provided for that option.) For other fragment navigations, including user-initiated ones, the navigation API state[p1053] is carried over from the previous entry.

    The classic history API state[p1053] is never carried over.
7. Let *entryToReplace* be *navigable*'s active session history entry[p1035] if *historyHandling* is "`replace`[p1062]", otherwise null.
8. If *entryToReplace* is non-null, then set *historyEntry*'s navigation API key[p1053] to *entryToReplace*'s navigation API key[p1053].
9. Let *history* be *navigable*'s active document[p1036]'s history object[p988].
10. Let *scriptHistoryIndex* be *history*'s index[p988].
11. Let *scriptHistoryLength* be *history*'s length[p988].
12. If *historyHandling* is "`push`[p1062]":
    1. Set *history*'s state[p988] to null.
    2. Increment *scriptHistoryIndex*.
    3. Set *scriptHistoryLength* to *scriptHistoryIndex* + 1.
13. Set *navigable*'s active document[p1036]'s URL to *url*.

14. Set *navigable*'s active session history entry[p1035] to *historyEntry*.

15. Update document for history step application[p1100] given *navigable*'s active document[p1036], *historyEntry*, true, *scriptHistoryIndex*, *scriptHistoryLength*, and *historyHandling*.

    Note

    This algorithm will be called twice as a result of a single fragment navigation: once synchronously, where best-guess values scriptHistoryIndex *and* scriptHistoryLength *are set,* `history.state`[p989] *is nulled out, and various events are fired; and once asynchronously, where the final values for index and length are set,* `history.state`[p989] *remains untouched, and no events are fired.*

16. Scroll to the fragment[p1104] given *navigable*'s active document[p1036].

    Note

    If the scrolling fails because the `Document`[p135] *is new and the relevant ID has not yet been parsed, then the second asynchronous call to update document for history step application[p1100] will take care of scrolling.*

17. Let *traversable* be *navigable*'s traversable navigable[p1037].

18. Append the following session history synchronous navigation steps[p1056] involving *navigable* to *traversable*:

    1. Finalize a same-document navigation[p1072] given *traversable*, *navigable*, *historyEntry*, *entryToReplace*, *historyHandling*, and *userInvolvement*.

    2. Invoke WebDriver BiDi fragment navigated with *navigable* and a new WebDriver BiDi navigation status whose id is *navigationId*, url is *url*, and status is "`complete`".

To **finalize a same-document navigation** given a traversable navigable[p1037] *traversable*, a navigable[p1035] *targetNavigable*, a session history entry[p1053] *targetEntry*, a session history entry[p1053]-or-null *entryToReplace*, a history handling behavior[p1062] *historyHandling*, and a user navigation involvement[p1062] *userInvolvement*:

Note

This is used by both fragment navigations[p1071] and by the URL and history update steps[p1078], which are the only synchronous updates to session history. By virtue of being synchronous, those algorithms are performed outside of the top-level traversable[p1037]'s session history traversal queue[p1037]. This puts them out of sync with the top-level traversable[p1037]'s current session history step[p1037], so this algorithm is used to resolve conflicts due to race conditions.

1. Assert: this is running on *traversable*'s session history traversal queue[p1037].

2. If *targetNavigable*'s active session history entry[p1035] is not *targetEntry*, then return.

3. Let *targetStep* be null.

4. Let *targetEntries* be the result of getting session history entries[p1058] for *targetNavigable*.

5. If *entryToReplace* is null:

    1. Clear the forward session history[p1059] of *traversable*.

    2. Set *targetStep* to *traversable*'s current session history step[p1037] + 1.

    3. Set *targetEntry*'s step[p1053] to *targetStep*.

    4. Append *targetEntry* to *targetEntries*.

    Otherwise:

    1. Replace *entryToReplace* with *targetEntry* in *targetEntries*.

    2. Set *targetEntry*'s step[p1053] to *entryToReplace*'s step[p1053].

    3. Set *targetStep* to *traversable*'s current session history step[p1037].

6. Apply the push/replace history step[p1090] *targetStep* to *traversable* given *historyHandling* and *userInvolvement*.

Note

This is done even for "replace[p1062]" navigations, as it resolves race conditions across multiple synchronous navigations.

7.4.2.3.4 Non-fetch schemes and external software §[p1073]

The input to attempt to create a non-fetch scheme document[p1073] is the **non-fetch scheme navigation params** struct. It is a lightweight version of navigation params[p1061] which only carries parameters relevant to the non-fetch scheme navigation case. It has the following items:

id

null or a navigation ID[p1062]

navigable

the navigable[p1035] experiencing the navigation

URL

a URL

target snapshot sandboxing flags

the target snapshot params[p1061]'s sandboxing flags[p1061] present during navigation

source snapshot has transient activation

a copy of the source snapshot params[p1060]'s has transient activation[p1060] boolean present during activation

initiator origin

an origin[p939] possibly for use in a user-facing prompt to confirm the invocation of an external software package

Note

This differs slightly from a document state[p1053]'s initiator origin[p1054] in that a non-fetch scheme navigation params[p1073]'s initiator origin[p1073] follows redirects up to the last fetch scheme URL in a redirect chain that ends in a non-fetch scheme URL.

navigation timing type

a `NavigationTimingType` used for creating the navigation timing entry for the new `Document`[p135] (if one is created)

user involvement

a user navigation involvement[p1062] used when obtaining a browsing context[p949] for the new `Document`[p135] (if one is created)

To **attempt to create a non-fetch scheme document**, given a non-fetch scheme navigation params[p1073] *navigationParams*:

1. Let *url* be *navigationParams*'s URL[p1073].
2. Let *navigable* be *navigationParams*'s navigable[p1073].
3. If *url* is to be handled using a mechanism that does not affect *navigable*, e.g., because *url*'s scheme is handled externally:
    1. Hand-off to external software[p1073] given *url*, *navigable*, *navigationParams*'s target snapshot sandboxing flags[p1073], *navigationParams*'s source snapshot has transient activation[p1073], and *navigationParams*'s initiator origin[p1073].
    2. Return null.
4. Handle *url* 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[p1266] for the given scheme. Return the result of displaying the inline content[p1112] given *navigable*, *navigationParams*'s id[p1073], *navigationParams*'s navigation timing type[p1073], and *navigationParams*'s user involvement[p1073].

Note

In the case of a registered handler being used, navigate[p1063] will be invoked with a new URL.

To **hand-off to external software** given a URL or response *resource*, a navigable[p1035] *navigable*, a sandboxing flag set[p957] *sandboxFlags*, a boolean *hasTransientActivation*, and an origin[p939] *initiatorOrigin*, user agents should:

1. If all of the following are true:

- *navigable* is not a top-level traversable[p1037];
- *sandboxFlags* has its sandboxed custom protocols navigation browsing context flag[p958] set; and
- *sandboxFlags* has its sandboxed top-level navigation with user activation browsing context flag[p957] set, or *hasTransientActivation* is false,

then return without invoking the external software package.

Note

Navigation inside an iframe toward external software can be seen by users as a new popup or a new top-level navigation. That's why its is allowed in sandboxed `iframe`[p404] *only when one of* `allow-popups`[p958], `allow-top-navigation`[p958], `allow-top-navigation-by-user-activation`[p958], *or* `allow-top-navigation-to-custom-protocols`[p959] *is specified.*

2. Perform the appropriate handoff of *resource* while attempting to mitigate the risk that this is an attempt to exploit the target software. For example, user agents could prompt the user to confirm that *initiatorOrigin* is to be allowed to invoke the external software in question. In particular, if *hasTransientActivation* is false, then the user agent should not invoke the external software package without prior user confirmation.

Example

For example, there could be a vulnerability in the target software's URL handler which a hostile page would attempt to exploit by tricking a user into clicking a link.

7.4.2.4 Preventing navigation §[p1074]

A couple of scenarios can intervene early in the navigation process and put the whole thing to a halt. This can be especially exciting when multiple navigables[p1035] are navigating at the same time, due to a session history traversal.

A navigable[p1035] *source* is **allowed by sandboxing to navigate** a second navigable[p1035] *target*, given a source snapshot params[p1060] *sourceSnapshotParams*, if the following steps return true:

1. If *source* is *target*, then return true.
2. If *source* is an ancestor of *target*, then return true.
3. If *target* is an ancestor of *source*:
    1. If *target* is not a top-level traversable[p1037], then return true.
    2. If *sourceSnapshotParams*'s has transient activation[p1060] is true, and *sourceSnapshotParams*'s sandboxing flags[p1060]'s sandboxed top-level navigation with user activation browsing context flag[p957] is set, then return false.
    3. If *sourceSnapshotParams*'s has transient activation[p1060] is false, and *sourceSnapshotParams*'s sandboxing flags[p1060]'s sandboxed top-level navigation without user activation browsing context flag[p957] is set, then return false.
    4. Return true.
4. If *target* is a top-level traversable[p1037]:
    1. If *source* is the one permitted sandboxed navigator[p957] of *target*, then return true.
    2. If *sourceSnapshotParams*'s sandboxing flags[p1060]'s sandboxed navigation browsing context flag[p957] is set, then return false.
    3. Return true.
5. If *sourceSnapshotParams*'s sandboxing flags[p1060]'s sandboxed navigation browsing context flag[p957] is set, then return false.
6. Return true.

To **check if unloading is canceled** for a list of navigables[p1035] *navigablesThatNeedBeforeUnload*, given an optional traversable navigable[p1037] *traversable*, an optional integer *targetStep*, and an optional user navigation involvement[p1062] *userInvolvementForNavigateEvent*, run these steps. They return "`canceled-by-beforeunload`", "`canceled-by-navigate`", or

"continue".

1. Let *documentsToFireBeforeunload* be the active document[p1036] of each item in *navigablesThatNeedBeforeUnload*.
2. Let *unloadPromptShown* be false.
3. Let *finalStatus* be "continue".
4. If *traversable* was given:
    1. Assert: *targetStep* and *userInvolvementForNavigateEvent* were given.
    2. Let *targetEntry* be the result of getting the target history entry[p1098] given *traversable* and *targetStep*.
    3. If *targetEntry* is not *traversable*'s current session history entry[p1035], and *targetEntry*'s document state[p1053]'s origin[p1054] is the same[p940] as *traversable*'s current session history entry[p1035]'s document state[p1053]'s origin[p1054]:

        Note

        In this case, we're going to fire the navigate[p1575] *event for* traversable *here. Because under some circumstances[p1021] it might be canceled, we need to do this separately from other traversal* navigate *events[p1092], which happen later.*

        Additionally, because we want beforeunload[p1574] *events to fire before* navigate[p1575] *events, this means we need to fire* beforeunload[p1574] *for* traversable *here (if applicable), instead of doing it as part of the below loop over* documentsToFireBeforeunload*.*

        1. Let *eventsFired* be false.
        2. Let *needsBeforeunload* be true if *navigablesThatNeedBeforeUnload* contains *traversable*; otherwise false.
        3. If *needsBeforeunload* is true, then remove *traversable*'s active document[p1036] from *documentsToFireBeforeunload*.
        4. Queue a global task[p1195] on the navigation and traversal task source[p1204] given *traversable*'s active window[p1036] to perform the following steps:
            1. If *needsBeforeunload* is true:
                1. Let (*unloadPromptShownForThisDocument*, *unloadPromptCanceledByThisDocument*) be the result of running the steps to fire beforeunload[p1076] given *traversable*'s active document[p1036] and false.
                2. If *unloadPromptShownForThisDocument* is true, then set *unloadPromptShown* to true.
                3. If *unloadPromptCanceledByThisDocument* is true, then set *finalStatus* to "canceled-by-beforeunload".
            2. If *finalStatus* is "canceled-by-beforeunload", then abort these steps.
            3. Let *navigation* be *traversable*'s active window[p1036]'s navigation API[p995].
            4. Let *navigateEventResult* be the result of firing a traverse navigate event[p1019] at *navigation* given *targetEntry* and *userInvolvementForNavigateEvent*.
            5. If *navigateEventResult* is false, then set *finalStatus* to "canceled-by-navigate".
            6. Set *eventsFired* to true.
        5. Wait until *eventsFired* is true.
        6. If *finalStatus* is not "continue", then return *finalStatus*.
5. Let *totalTasks* be the size of *documentsToFireBeforeunload*.
6. Let *completedTasks* be 0.
7. For each *document* of *documentsToFireBeforeunload*, queue a global task[p1195] on the navigation and traversal task source[p1204] given *document*'s relevant global object[p1151] to run the steps:

1. Let (*unloadPromptShownForThisDocument*, *unloadPromptCanceledByThisDocument*) be the result of running the steps to fire `beforeunload`[p1076] given *document* and *unloadPromptShown*.
2. If *unloadPromptShownForThisDocument* is true, then set *unloadPromptShown* to true.
3. If *unloadPromptCanceledByThisDocument* is true, then set *finalStatus* to "`canceled-by-beforeunload`".
4. Increment *completedTasks*.

8. Wait for *completedTasks* to be *totalTasks*.
9. Return *finalStatus*.

The **steps to fire `beforeunload`** given a `Document`[p135] *document* and a boolean *unloadPromptShown* are:

1. Let *unloadPromptCanceled* be false.
2. Increase the *document*'s unload counter[p1114] by 1.
3. Increase *document*'s relevant agent[p1141]'s event loop[p1193]'s termination nesting level[p1114] by 1.
4. Let *eventFiringResult* be the result of firing an event named `beforeunload`[p1574] at *document*'s relevant global object[p1151], using `BeforeUnloadEvent`[p1030], with the `cancelable` attribute initialized to true.
5. Decrease *document*'s relevant agent[p1141]'s event loop[p1193]'s termination nesting level[p1114] by 1.
6. If all of the following are true:
   - *unloadPromptShown* is false;
   - *document*'s active sandboxing flag set[p959] does not have its sandboxed modals flag[p958] set;
   - *document*'s relevant global object[p1151] has sticky activation[p868];
   - *eventFiringResult* is false, or the `returnValue`[p1030] attribute of *event* is not the empty string; and
   - showing an unload prompt is unlikely to be annoying, deceptive, or pointless,

   then:
   1. Set *unloadPromptShown* to true.
   2. Let *userPromptHandler* be the result of WebDriver BiDi user prompt opened with *document*'s relevant global object[p1151], "`beforeunload`", and "".
   3. If *userPromptHandler* is "`dismiss`", then set *unloadPromptCanceled* to true.
   4. If *userPromptHandler* is "`none`":
      1. Ask the user to confirm that they wish to unload the document, and pause[p1203] while waiting for the user's response.

         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`[p1030] attribute is ignored.

      2. If the user did not confirm the page navigation, then set *unloadPromptCanceled* to true.
   5. Invoke WebDriver BiDi user prompt closed with *document*'s relevant global object[p1151], "`beforeunload`", and true if *unloadPromptCanceled* is false or false otherwise.
7. Decrease *document*'s unload counter[p1114] by 1.
8. Return (*unloadPromptShown*, *unloadPromptCanceled*).

7.4.2.5 Aborting navigation §[p1076]

Each navigable[p1035] has an **ongoing navigation**, which is a navigation ID[p1062], "`traversal`", or null, initially null. It is used to track

navigation aborting and to prevent any navigations from taking place during traversal[p1090].

To **set the ongoing navigation** for a navigable[p1035] *navigable* to *newValue*:

1. If *navigable*'s ongoing navigation[p1076] is equal to *newValue*, then return.
2. Inform the navigation API about aborting navigation[p1010] given *navigable*.
3. Set *navigable*'s ongoing navigation[p1076] to *newValue*.

7.4.3 Reloading and traversing §[p1077]

To **reload** a navigable[p1035] *navigable* given an optional serialized state[p1053]-or-null ***navigationAPIState*** (default null), an optional user navigation involvement[p1062] ***userInvolvement*** (default "none[p1062]"), and an optional navigation API method tracker[p1007]-or-null ***apiMethodTracker*** (default null):

1. If *userInvolvement* is not "browser UI[p1062]":
    1. Let *navigation* be *navigable*'s active window[p1036]'s navigation API[p995].
    2. Let *destinationNavigationAPIState* be *navigable*'s active session history entry[p1035]'s navigation API state[p1053].
    3. If *navigationAPIState* is not null, then set *destinationNavigationAPIState* to *navigationAPIState*.
    4. Let *continue* be the result of firing a push/replace/reload navigate event[p1019] at *navigation* with *navigationType*[p1019] set to "reload[p997]", *isSameDocument*[p1019] set to false, *userInvolvement*[p1019] set to *userInvolvement*, *destinationURL*[p1019] set to *navigable*'s active session history entry[p1035]'s URL[p1053], *navigationAPIState*[p1019] set to *destinationNavigationAPIState*, and *apiMethodTracker*[p1019] set to *apiMethodTracker*.
    5. If *continue* is false, then return.
2. Set *navigable*'s active session history entry[p1035]'s document state[p1053]'s reload pending[p1055] to true.
3. Let *traversable* be *navigable*'s traversable navigable[p1037].
4. Append the following session history traversal steps[p1056] to *traversable*:
    1. Apply the reload history step[p1090] to *traversable* given *userInvolvement*.

To **traverse the history by a delta** given a traversable navigable[p1037] *traversable*, an integer *delta*, and an optional Document[p135] ***sourceDocument***:

1. Let *sourceSnapshotParams* and *initiatorToCheck* be null.
2. Let *userInvolvement* be "browser UI[p1062]".
3. If *sourceDocument* is given:
    1. Set *sourceSnapshotParams* to the result of snapshotting source snapshot params[p1060] given *sourceDocument*.
    2. Set *initiatorToCheck* to *sourceDocument*'s node navigable[p1036].
    3. Set *userInvolvement* to "none[p1062]".
4. Append the following session history traversal steps[p1056] to *traversable*:
    1. Let *allSteps* be the result of getting all used history steps[p1059] for *traversable*.
    2. Let *currentStepIndex* be the index of *traversable*'s current session history step[p1037] within *allSteps*.
    3. Let *targetStepIndex* be *currentStepIndex* plus *delta*.
    4. If *allSteps*[*targetStepIndex*] does not exist, then abort these steps.
    5. Apply the traverse history step[p1090] *allSteps*[*targetStepIndex*] to *traversable*, given *sourceSnapshotParams*, *initiatorToCheck*, and *userInvolvement*.

7.4.4 Non-fragment synchronous "navigations" §p1078

Apart from the navigate[p1063] algorithm, session history entries[p1053] can be pushed or replaced via one more mechanism, the URL and history update steps[p1078]. The most well-known callers of these steps are the `history.replaceState()`[p989] and `history.pushState()`[p989] APIs, but various other parts of the standard also need to perform updates to the active history entry[p1035], and they use these steps to do so.

The **URL and history update steps**, given a `Document`[p135] *document*, a URL *newURL*, an optional serialized state[p1053]-or-null ***serializedData*** (default null), and an optional history handling behavior[p1062] ***historyHandling*** (default "`replace`[p1062]"), are:

1. Let *navigable* be *document*'s node navigable[p1036].
2. Let *activeEntry* be *navigable*'s active session history entry[p1035].
3. Let *newEntry* be a new session history entry[p1053], with

   URL[p1053]
   newURL
   serialized state[p1053]
   if *serializedData* is not null, *serializedData*; otherwise *activeEntry*'s classic history API state[p1053]
   document state[p1053]
   activeEntry's document state[p1053]
   scroll restoration mode[p1053]
   activeEntry's scroll restoration mode[p1053]
   persisted user state[p1053]
   activeEntry's persisted user state[p1053]
4. If *document*'s is initial about:blank[p136] is true, then set *historyHandling* to "`replace`[p1062]".

   Note

   This means that `pushState()`[p989] on an initial about:blank[p136] `Document`[p135] behaves as a `replaceState()`[p989] call.
5. Let *entryToReplace* be *activeEntry* if *historyHandling* is "`replace`[p1062]", otherwise null.
6. If *entryToReplace* is non-null, then set *newEntry*'s navigation API key[p1053] to *entryToReplace*'s navigation API key[p1053].
7. If *historyHandling* is "`push`[p1062]":
   1. Increment *document*'s history object[p988]'s index[p988].
   2. Set *document*'s history object[p988]'s length[p988] to its index[p988] + 1.

   Note

   These are temporary best-guess values for immediate synchronous access.
8. If *serializedData* is not null, then restore the history object state[p1101] given *document* and *newEntry*.
9. Set the URL[p101] given *document* to *newURL*.

   Note

   Since this is neither a navigation[p1063] nor a history traversal[p1077], it does not cause a `hashchange`[p1575] event to be fired.
10. Set *document*'s latest entry[p1055] to *newEntry*.
11. Set *navigable*'s active session history entry[p1035] to *newEntry*.
12. Update the navigation API entries for a same-document navigation[p998] given *document*'s relevant global object[p1151]'s navigation API[p995], *newEntry*, and *historyHandling*.
13. Let *traversable* be *navigable*'s traversable navigable[p1037].
14. Append the following session history synchronous navigation steps[p1056] involving *navigable* to *traversable*:
    1. Finalize a same-document navigation[p1072] given *traversable*, *navigable*, *newEntry*, *entryToReplace*, *historyHandling*, and "`none`[p1062]".
    2. Invoke WebDriver BiDi history updated with *navigable*.

Note

Although both fragment navigation[p1071] and the URL and history update steps[p1078] perform synchronous history updates, only fragment navigation contains a synchronous call to update document for history step application[p1100]. The URL and history update steps[p1078] instead perform a few select updates inside the above algorithm, omitting others. This is somewhat of an unfortunate historical accident, and generally leads to web-developer sadness about the inconsistency. For example, this means that `popstate`[p1576] *events fire for fragment navigations, but not for* `history.pushState()`[p989] *calls.*

7.4.5 Populating a session history entry §[p1079]

As explained in the overview[p1052], both navigation[p1059] and traversal[p1077] involve creating a session history entry[p1053] and then attempting to populate its document[p1053] member, so that it can be presented inside the navigable[p1035].

This involves either: using an already-given response[p1066]; using the srcdoc resource[p1054] stored in the session history entry[p1053]; or fetching[p1082]. The process has several failure modes, which can either result in doing nothing (leaving the navigable[p1035] on its currently-active[p1036] `Document`[p135]) or can result in populating the session history entry[p1053] with an error document[p1112].

To **attempt to populate the history entry's document** for a session history entry[p1053] *entry*, given a navigable[p1035] *navigable*, a `NavigationTimingType` *navTimingType*, a source snapshot params[p1060] *sourceSnapshotParams*, a target snapshot params[p1061] *targetSnapshotParams*, a user navigation involvement[p1062] *userInvolvement*, an optional navigation ID[p1062]-or-null *navigationId* (default null), an optional navigation params[p1061]-or-null *navigationParams* (default null), an optional string *cspNavigationType* (default "`other`"), an optional boolean ***allowPOST*** (default false), and optional algorithm steps ***completionSteps*** (default an empty algorithm):

1. Assert: this is running in parallel[p44].
2. Assert: if *navigationParams* is non-null, then *navigationParams*'s response[p1061] is non-null.
3. Let *documentResource* be *entry*'s document state[p1053]'s resource[p1054].
4. If *navigationParams* is null:
    1. If *documentResource* is a string, then set *navigationParams* to the result of creating navigation params from a srcdoc resource[p1081] given *entry*, *navigable*, *targetSnapshotParams*, *userInvolvement*, *navigationId*, and *navTimingType*.
    2. Otherwise, if all of the following are true:
        - *entry*'s URL[p1053]'s scheme is a fetch scheme; and
        - *documentResource* is null, or *allowPOST* is true and *documentResource*'s request body[p1055] is not failure,

        then set *navigationParams* to the result of creating navigation params by fetching[p1082] given *entry*, *navigable*, *sourceSnapshotParams*, *targetSnapshotParams*, *cspNavigationType*, *userInvolvement*, *navigationId*, and *navTimingType*.
    3. Otherwise, if *entry*'s URL[p1053]'s scheme is not a fetch scheme, then set *navigationParams* to a new non-fetch scheme navigation params[p1073], with

        id[p1073]
        : *navigationId*

        navigable[p1073]
        : *navigable*

        URL[p1073]
        : *entry*'s URL[p1053]

        target snapshot sandboxing flags[p1073]
        : *targetSnapshotParams*'s sandboxing flags[p1061]

        source snapshot has transient activation[p1073]
        : *sourceSnapshotParams*'s has transient activation[p1060]

        initiator origin[p1073]
        : *entry*'s document state[p1053]'s initiator origin[p1054]

        navigation timing type[p1073]
        : *navTimingType*

user involvement[p1073]
userInvolvement

5. Queue a global task[p1195] on the navigation and traversal task source[p1204], given *navigable*'s active window[p1046], to run these steps:

    1. If *navigable*'s ongoing navigation[p1076] no longer equals *navigationId*, then run *completionSteps* and abort these steps.

    2. Let *saveExtraDocumentState* be true.

    > **Note**
    > *Usually, in the cases where we end up populating* entry's *document state*[p1053]'s *document*[p1054]*, we then want to save some of the state from that* `Document`[p135] *into* entry*. This ensures that if there are future traversals to* entry *where its document*[p1054] *has been destroyed*[p1054]*, we can use that state when creating a new* `Document`[p135]*.*
    >
    > *However, in some specific cases, saving the state would be unhelpful. For those, we set* saveExtraDocumentState *to false later in this algorithm.*

    3. If *navigationParams* is a non-fetch scheme navigation params[p1073]:

        1. Set *entry*'s document state[p1053]'s document[p1054] to the result of running attempt to create a non-fetch scheme document[p1073] given *navigationParams*.

        > **Note**
        > *This can result in setting* entry's *document state*[p1053]'s *document*[p1054] *to null, e.g., when handing-off to external software*[p1073].

        2. Set *saveExtraDocumentState* to false.

    4. Otherwise, if any of the following are true:

        - *navigationParams* is null;
        - the result of should navigation response to navigation request of type in target be blocked by Content Security Policy? given *navigationParams*'s request[p1061], *navigationParams*'s response[p1061], *navigationParams*'s policy container[p1061]'s CSP list[p960], *cspNavigationType*, and *navigable* is "`Blocked`";
        - *navigationParams*'s reserved environment[p1061] is non-null and the result of checking a navigation response's adherence to its embedder policy[p956] given *navigationParams*'s response[p1061], *navigable*, and *navigationParams*'s policy container[p1061]'s embedder policy[p960] is false; or
        - the result of checking a navigation response's adherence to `X-Frame-Options`[p1136] given *navigationParams*'s response[p1061], *navigable*, *navigationParams*'s policy container[p1061]'s CSP list[p960], and *navigationParams*'s origin[p1061] is false,

        then:

        1. Set *entry*'s document state[p1053]'s document[p1054] to the result of creating a document for inline content that doesn't have a DOM[p1112], given *navigable*, null, *navTimingType*, and *userInvolvement*. The inline content should indicate to the user the sort of error that occurred.

        2. Make document unsalvageable[p1102] given *entry*'s document state[p1053]'s document[p1054] and "`navigation-failure`[p1031]".

        3. Set *saveExtraDocumentState* to false.

        4. If *navigationParams* is not null:

            1. Run the environment discarding steps[p1144] for *navigationParams*'s reserved environment[p1061].

            2. Invoke WebDriver BiDi navigation failed with *navigable* and a new WebDriver BiDi navigation status whose id is *navigationId*, status is "`canceled`", and url is *navigationParams*'s response[p1061]'s URL.

    5. Otherwise, if *navigationParams*'s response[p1061] has a `` `Content-Disposition` `` header specifying the `attachment`

disposition type:

1. Let *sourceAllowsDownloading* be *sourceSnapshotParams*'s allows downloading[p1060].
2. Let *targetAllowsDownloading* be false if *navigationParams*'s final sandboxing flag set[p1061] has the sandboxed downloads browsing context flag[p958] set; otherwise true.
3. Let *uaAllowsDownloading* be true.
4. Optionally, the user agent may set *uaAllowsDownloading* to false, if it believes doing so would safeguard the user from a potentially hostile download.
5. If *sourceAllowsDownloading*, *targetAllowsDownloading*, and *uaAllowsDownloading* are true:
    1. Handle as a download[p323] *navigationParams*'s response[p1061] with *navigable* and *navigationId*.

Note

This branch leaves entry's *document state*[p1053]'s *document*[p1054] *as null.*

6. Otherwise, if *navigationParams*'s response[p1061]'s status is not 204 and is not 205, then set *entry*'s document state[p1053]'s document[p1054] to the result of loading a document[p1088] given *navigationParams*, *sourceSnapshotParams*, and *entry*'s document state[p1053]'s initiator origin[p1054].

Note

This can result in setting entry's *document state*[p1053]'s *document*[p1054] *to null, e.g., when handing-off to external software*[p1073].

7. If *entry*'s document state[p1053]'s document[p1054] is not null:
    1. Set *entry*'s document state[p1053]'s ever populated[p1055] to true.
    2. If *saveExtraDocumentState* is true:
        1. Let *document* be *entry*'s document state[p1053]'s document[p1054].
        2. Set *entry*'s document state[p1053]'s origin[p1054] to *document*'s origin.
        3. If *document*'s URL requires storing the policy container in history[p960]:
            1. Assert: *navigationParams* is a navigation params[p1061] (i.e., neither null nor a non-fetch scheme navigation params[p1073]).
            2. Set *entry*'s document state[p1053]'s history policy container[p1054] to *navigationParams*'s policy container[p1061].
    3. If *entry*'s document state[p1053]'s request referrer[p1054] is "`client`", and *navigationParams* is a navigation params[p1061] (i.e., neither null nor a non-fetch scheme navigation params[p1073]):
        1. Assert: *navigationParams*'s request[p1061] is not null.
        2. Set *entry*'s document state[p1053]'s request referrer[p1054] to *navigationParams*'s request[p1061]'s referrer.
8. Run *completionSteps*.

To **create navigation params from a srcdoc resource** given a session history entry[p1053] *entry*, a navigable[p1035] *navigable*, a target snapshot params[p1061] *targetSnapshotParams*, a user navigation involvement[p1062] *userInvolvement*, a navigation ID[p1062]-or-null *navigationId*, and a `NavigationTimingType` *navTimingType*:

1. Let *documentResource* be *entry*'s document state[p1053]'s resource[p1054].
2. Let *response* be a new response with

    URL
    `about:srcdoc`[p100]
    header list
    « (`` `Content-Type`[p102] ``, `` `text/html` ``) »
    body
    the UTF-8 encoding of *documentResource*, as a body

3. Let *responseOrigin* be the result of determining the origin[p1049] given *response*'s URL, *targetSnapshotParams*'s sandboxing flags[p1061], and *entry*'s document state[p1053]'s origin[p1054].

4. Let *coop* be a new opener policy[p945].

5. Let *coopEnforcementResult* be a new opener policy enforcement result[p948] with

   url[p948]
   response's URL
   origin[p948]
   responseOrigin
   opener policy[p948]
   coop

6. Let *policyContainer* be the result of determining navigation params policy container[p961] given *response*'s URL, *entry*'s document state[p1053]'s history policy container[p1054], null, *navigable*'s container document[p1038]'s policy container[p136], and null.

7. Return a new navigation params[p1061], with

   id[p1061]
   navigationId
   navigable[p1061]
   navigable
   request[p1061]
   null
   response[p1061]
   response
   fetch controller[p1061]
   null
   commit early hints[p1061]
   null
   COOP enforcement result[p1061]
   coopEnforcementResult
   reserved environment[p1061]
   null
   origin[p1061]
   responseOrigin
   policy container[p1061]
   policyContainer
   final sandboxing flag set[p1061]
   targetSnapshotParams's sandboxing flags[p1061]
   `iframe` element referrer policy[p1061]
   targetSnapshotParams's `iframe` element referrer policy[p1061]
   opener policy[p1061]
   coop
   navigation timing type[p1062]
   navTimingType
   about base URL[p1062]
   entry's document state[p1053]'s about base URL[p1054]
   user involvement[p1062]
   userInvolvement

To **create navigation params by fetching** given a session history entry[p1053] *entry*, a navigable[p1035] *navigable*, a source snapshot params[p1060] *sourceSnapshotParams*, a target snapshot params[p1061] *targetSnapshotParams*, a string *cspNavigationType*, a user navigation involvement[p1062] *userInvolvement*, a navigation ID[p1062]-or-null *navigationId*, and a `NavigationTimingType` *navTimingType*, perform the following steps. They return a navigation params[p1061], a non-fetch scheme navigation params[p1073], or null.

Note

This algorithm mutates entry.

1. Assert: this is running in parallel[p44].

2. Let *documentResource* be *entry*'s document state[p1053]'s resource[p1054].

3. Let *request* be a new request, with

URL
entry's URL[p1053]

client
sourceSnapshotParams's fetch client[p1060]

destination
"`document`" Note *The destination is updated below when* navigable *has a container*[p1038].

credentials mode
"`include`"

use-URL-credentials flag
set

redirect mode
"`manual`"

replaces client id
navigable's active document[p1036]'s relevant settings object[p1151]'s id[p1143]

mode
"`navigate`"

referrer
entry's document state[p1053]'s request referrer[p1054]

referrer policy
entry's document state[p1053]'s request referrer policy[p1054]

policy container
sourceSnapshotParams's source policy container[p1060]

traversable for user prompts
navigable's top-level traversable[p1037]

4. If *navigable* is a top-level traversable[p1037], then set *request*'s top-level navigation initiator origin to *entry*'s document state[p1053]'s initiator origin[p1054].

5. If *request*'s client is null:

   Note

   This only occurs[p1063] *in the case of a browser UI-initiated navigation.*

   1. Set *request*'s origin to a new opaque origin[p939].
   2. Set *request*'s service-workers mode to "`all`".
   3. Set *request*'s referrer to "`no-referrer`".

6. If *documentResource* is a POST resource[p1055]:

   1. Set *request*'s method to `` `POST` ``.
   2. Set *request*'s body to *documentResource*'s request body[p1055].
   3. Set `` `Content-Type` `` to *documentResource*'s request content-type[p1055] in *request*'s header list.

7. If *entry*'s document state[p1053]'s reload pending[p1055] is true, then set *request*'s reload-navigation flag.

8. Otherwise, if *entry*'s document state[p1053]'s ever populated[p1055] is true, then set *request*'s history-navigation flag.

9. If *sourceSnapshotParams*'s has transient activation[p1060] is true, then set *request*'s user-activation to true.

10. If *navigable*'s container[p1038] is non-null:

    1. If the *navigable*'s container[p1038] has a browsing context scope origin[p1088], then set *request*'s origin to that browsing context scope origin[p1088].
    2. Set *request*'s destination to *navigable*'s container[p1038]'s local name.
    3. If *sourceSnapshotParams*'s fetch client[p1060] is *navigable*'s container document[p1038]'s relevant settings object[p1151], then set *request*'s initiator type to *navigable*'s container[p1038]'s local name.

       Note

       This ensure that only container-initiated navigations are reported to resource timing.

11. Let *response* be null.

12. Let *responseOrigin* be null.

13. Let *fetchController* be null.

14. Let *coopEnforcementResult* be a new opener policy enforcement result[p948], with

    url[p948]
    : *navigable*'s active document[p1036]'s URL

    origin[p948]
    : *navigable*'s active document[p1036]'s origin

    opener policy[p948]
    : *navigable*'s active document[p1036]'s opener policy[p136]

    current context is navigation source[p948]
    : true if *navigable*'s active document[p1036]'s origin is same origin[p940] with *entry*'s document state[p1053]'s initiator origin[p1054]
      otherwise false

15. Let *finalSandboxFlags* be an empty sandboxing flag set[p957].

16. Let *responsePolicyContainer* be null.

17. Let *responseCOOP* be a new opener policy[p945].

18. Let *locationURL* be null.

19. Let *currentURL* be *request*'s current URL.

20. Let *commitEarlyHints* be null.

21. While true:

    1. If *request*'s reserved client is not null and *currentURL*'s origin is not the same[p940] as *request*'s reserved client's creation URL[p1143]'s origin:

       1. Run the environment discarding steps[p1144] for *request*'s reserved client.

       2. Set *request*'s reserved client to null.

       3. Set *commitEarlyHints* to null.

       Note

       Preloaded links from early hint headers[p194] remain in the preload cache after a same origin[p940] redirect, but get discarded when the redirect is cross-origin.

    2. If *request*'s reserved client is null:

       1. Let *topLevelCreationURL* be *currentURL*.

       2. Let *topLevelOrigin* be null.

       3. If *navigable* is not a top-level traversable[p1037]:

          1. Let *parentEnvironment* be *navigable*'s parent[p1035]'s active document[p1036]'s relevant settings object[p1151].

          2. Set *topLevelCreationURL* to *parentEnvironment*'s top-level creation URL[p1144].

          3. Set *topLevelOrigin* to *parentEnvironment*'s top-level origin[p1144].

       4. Set *request*'s reserved client to a new environment[p1143] whose id[p1143] is a unique opaque string, target browsing context[p1144] is *navigable*'s active browsing context[p1036], creation URL[p1143] is *currentURL*, top-level creation URL[p1144] is *topLevelCreationURL*, and top-level origin[p1144] is *topLevelOrigin*.

       Note

       The created environment's active service worker[p1144] is set in the Handle Fetch algorithm during the fetch if the request URL matches a service worker registration. [SW][p1587]

    3. If the result of should navigation request of type be blocked by Content Security Policy? given *request* and *cspNavigationType* is "`Blocked`", then set *response* to a network error and break. [CSP][p1580]

4. Set *response* to null.

5. If *fetchController* is null, then set *fetchController* to the result of fetching *request*, with *processEarlyHintsResponse* set to *processEarlyHintsResponse* as defined below, *processResponse* set to *processResponse* as defined below, and *useParallelQueue* set to true.

   Let *processEarlyHintsResponse* be the following algorithm given a response *earlyResponse*:

   1. If *commitEarlyHints* is null, then set *commitEarlyHints* to the result of processing early hint headers[p195] given *earlyResponse* and *request*'s reserved client.

   Let *processResponse* be the following algorithm given a response *fetchedResponse*:

   1. Set *response* to *fetchedResponse*.

6. Otherwise, process the next manual redirect for *fetchController*.

   Note

   This will result in calling the processResponse *we supplied above, during our first iteration through the loop, and thus setting* response.

   Note

   Navigation handles redirects manually as navigation is the only place in the web platform that cares for redirects to `mailto:` *URLs and such.*

7. Wait until either *response* is non-null, or *navigable*'s ongoing navigation[p1076] changes to no longer equal *navigationId*.

   If the latter condition occurs, then abort *fetchController*, and return.

   Otherwise, proceed onward.

8. If *request*'s body is null, then set *entry*'s document state[p1053]'s resource[p1054] to null.

   Note

   Fetch unsets the body for particular redirects.

9. Set *responsePolicyContainer* to the result of creating a policy container from a fetch response[p960] given *response* and *request*'s reserved client.

10. Set *finalSandboxFlags* to the union of *targetSnapshotParams*'s sandboxing flags[p1061] and *responsePolicyContainer*'s CSP list[p960]'s CSP-derived sandboxing flags[p959].

11. Set *responseOrigin* to the result of determining the origin[p1049] given *response*'s URL, *finalSandboxFlags*, and *entry*'s document state[p1053]'s initiator origin[p1054].

    Note

    If response *is a redirect, then* response's *URL will be the URL that led to the redirect to* response's *location URL; it will not be the location URL itself.*

12. If *navigable* is a top-level traversable[p1037]:

    1. Set *responseCOOP* to the result of obtaining an opener policy[p946] given *response* and *request*'s reserved client.

    2. Set *coopEnforcementResult* to the result of enforcing the response's opener policy[p948] given *navigable*'s active browsing context[p1036], *response*'s URL, *responseOrigin*, *responseCOOP*, *coopEnforcementResult*, and *request*'s referrer.

    3. If *finalSandboxFlags* is not empty and *responseCOOP*'s value[p945] is not "`unsafe-none`[p945]", then set *response* to an appropriate network error and break.

       Note

       This results in a network error as one cannot simultaneously provide a clean slate to a response using opener policy and sandbox the result of navigating to that response.

13. If *response* is not a network error, *navigable* is a child navigable[p1039], and the result of performing a cross-origin resource policy check with *navigable*'s container document[p1038]'s origin, *navigable*'s container document[p1038]'s relevant settings object[p1151], *request*'s destination, *response*, and true is **blocked**, then set *response* to a network error and break.

    Note

    Here we're running the cross-origin resource policy check against the parent navigable[p1035] rather than navigable *itself. This is because we care about the same-originness of the embedded content against the parent context, not the navigation source.*

14. Set *locationURL* to *response*'s location URL given *currentURL*'s fragment.

15. If *locationURL* is failure or null, then break.

16. Assert: *locationURL* is a URL.

17. Set *entry*'s classic history API state[p1053] to StructuredSerializeForStorage[p128](null).

18. Let *oldDocState* be *entry*'s document state[p1053].

19. Set *entry*'s document state[p1053] to a new document state[p1054], with

    history policy container[p1054]
    : a clone[p960] of the *oldDocState*'s history policy container[p1054] if it is non-null; null otherwise

    request referrer[p1054]
    : *oldDocState*'s request referrer[p1054]

    request referrer policy[p1054]
    : *oldDocState*'s request referrer policy[p1054]

    initiator origin[p1054]
    : *oldDocState*'s initiator origin[p1054]

    origin[p1054]
    : *oldDocState*'s origin[p1054]

    about base URL[p1054]
    : *oldDocState*'s about base URL[p1054]

    resource[p1054]
    : *oldDocState*'s resource[p1054]

    ever populated[p1055]
    : *oldDocState*'s ever populated[p1055]

    navigable target name[p1055]
    : *oldDocState*'s navigable target name[p1055]

    Note

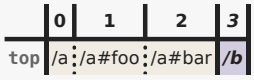
    For the navigation case, only entry *referenced* oldDocState*, which was created early in the navigate algorithm[p1065]. So for navigations, this is functionally just an update to* entry*'s document state[p1053]. For the traversal case, it's possible adjacent session history entries[p1053] also reference* oldDocState*, in which case they will continue doing so even after we've updated* entry*'s document state[p1053].*

    Note

    oldDocState*'s history policy container[p1054] is only ever non-null here in the traversal case, after we've populated it during a navigation to a URL that requires storing the policy container in history[p960].*

    Example

    The setup is given by the following Jake diagram[p1040]:

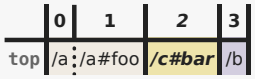
    

    Also assume that the document state[p1053] shared by the entries in steps 0, 1, and 2 has a null document[p1054], i.e., bfcache[p1054] is not in play.

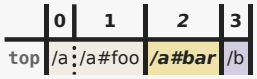
    Now consider the scenario where we traverse back to step 2, but this time when fetching `/a`, the server responds with a `Location` header pointing to `/c`. That is, *locationURL* points to `/c` and so we have reached

this step instead of breaking out of the loop.

In this case, we replace the document state[p1053] of the session history entry[p1053] occupying step 2, but we do *not* replace the document state of the entries occupying steps 0 and 1. The resulting Jake diagram[p1040] looks like this:



Note that we perform this replacement even if we end up in a redirect chain back to the original URL, for example if `/c` itself had a `Location` header pointing to `/a`. Such a case would end up like so:



20. If *locationURL*'s scheme is not an HTTP(S) scheme:

    1. Set *entry*'s document state[p1053]'s resource[p1054] to null.
    2. Break.

21. Set *currentURL* to *locationURL*.

22. Set *entry*'s URL[p1053] to *currentURL*.

Note

By the end of this loop we will be in one of these scenarios:

- locationURL *is failure, because of an unparseable* `Location` *header.*
- locationURL *is null, either because* response *is a network error or because we successfully fetched a non-network error HTTP(S) response with no* `Location` *header.*
- locationURL *is a URL with a non-HTTP(S) scheme.*

22. If *locationURL* is a URL whose scheme is not a fetch scheme, then return a new non-fetch scheme navigation params[p1073], with

id[p1073]
: *navigationId*

navigable[p1073]
: *navigable*

URL[p1073]
: *locationURL*

target snapshot sandboxing flags[p1073]
: *targetSnapshotParams*'s sandboxing flags[p1061]

source snapshot has transient activation[p1073]
: *sourceSnapshotParams*'s has transient activation[p1060]

initiator origin[p1073]
: *responseOrigin*

navigation timing type[p1073]
: *navTimingType*

user involvement[p1073]
: *userInvolvement*

Note

At this point, request's *current URL is the last URL in the redirect chain with a fetch scheme before redirecting to a non-fetch scheme URL. It is this URL's origin that will be used as the initiator origin for navigations to non-fetch scheme URLs.*

23. If any of the following are true:

    - *response* is a network error;

- *locationURL* is failure; or
- *locationURL* is a URL whose scheme is a fetch scheme,

then return null.

> **Note**
> *We allow redirects to non-fetch scheme URLs, but redirects to fetch scheme URLs that aren't HTTP(S) are treated like network errors.*

24. Assert: *locationURL* is null and *response* is not a network error.

25. Let *resultPolicyContainer* be the result of determining navigation params policy container[p961] given *response*'s URL, *entry*'s document state[p1053]'s history policy container[p1054], *sourceSnapshotParams*'s source policy container[p1060], null, and *responsePolicyContainer*.

26. If *navigable*'s container[p1038] is an `iframe`[p404], and *response*'s timing allow passed flag is set, then set *navigable*'s container[p1038]'s pending resource-timing start time[p409] to null.

> **Note**
> *If the `iframe`[p404] is allowed to report to resource timing, we don't need to run its fallback steps as the normal reporting would happen.*

27. Return a new navigation params[p1061], with

id[p1061]
: *navigationId*
navigable[p1061]
: *navigable*
request[p1061]
: *request*
response[p1061]
: *response*
fetch controller[p1061]
: *fetchController*
commit early hints[p1061]
: *commitEarlyHints*
opener policy[p1061]
: *responseCOOP*
reserved environment[p1061]
: *request*'s reserved client
origin[p1061]
: *responseOrigin*
policy container[p1061]
: *resultPolicyContainer*
final sandboxing flag set[p1061]
: *finalSandboxFlags*
COOP enforcement result[p1061]
: *coopEnforcementResult*
navigation timing type[p1062]
: *navTimingType*
about base URL[p1062]
: *entry*'s document state[p1053]'s about base URL[p1054]
user involvement[p1062]
: *userInvolvement*

An element has a **browsing context scope origin** if its `Document`[p135]'s node navigable[p1036] is a top-level traversable[p1037] or if all of its `Document`[p135]'s ancestor navigables[p1041] have active documents[p1036] whose origins are the same origin[p940] as the element's node document's origin. If an element has a browsing context scope origin[p1088], then its value is the origin of the element's node document.

This definition is broken and needs investigation to see what it was intended to express: see issue #4703.

To **load a document** given navigation params[p1061] *navigationParams*, source snapshot params[p1060] *sourceSnapshotParams*, and

origin[p939] *initiatorOrigin*, perform the following steps. They return a Document[p135] or null.

1. Let *type* be the computed type of *navigationParams*'s response[p1061].

2. If the user agent has been configured to process resources of the given *type* using some mechanism other than rendering the content in a navigable[p1035], then skip this step. Otherwise, if the *type* is one of the following types:

   ↪ **an HTML MIME type**

   Return the result of loading an HTML document[p1110], given *navigationParams*.

   ↪ **an XML MIME type that is not an explicitly supported XML MIME type**[p1089]

   Return the result of loading an XML document[p1110] given *navigationParams* and *type*.

   ↪ **a JavaScript MIME type**

   ↪ **a JSON MIME type that is not an explicitly supported JSON MIME type**[p1089]

   ↪ "**text/css**[p1577]"

   ↪ "**text/plain**"

   ↪ "**text/vtt**[p1578]"

   Return the result of loading a text document[p1111] given *navigationParams* and *type*.

   ↪ "**multipart/x-mixed-replace**[p1547]"

   Return the result of loading a `multipart/x-mixed-replace` document[p1111], given *navigationParams*, *sourceSnapshotParams*, and *initiatorOrigin*.

   ↪ **a supported image, video, or audio type**

   Return the result of loading a media document[p1112] given *navigationParams* and *type*.

   ↪ "**application/pdf**"

   ↪ "**text/pdf**"

   If the user agent's PDF viewer supported[p1270] is true, return the result of creating a document for inline content that doesn't have a DOM[p1112] given *navigationParams*'s navigable[p1061], *navigationParams*'s id[p1061], *navigationParams*'s navigation timing type[p1062], and *navigationParams*'s user involvement[p1062].

   Otherwise, proceed onward.

   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, or for which the user agent has dedicated processing rules. For example, a web browser with a built-in Atom feed viewer would be said to explicitly support the `application/atom+xml`[p1577] MIME type.

   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, or for which the user agent has dedicated processing rules.

   Note

   In both cases, the external application or user agent will either display the content inline[p1112] *directly in* navigationParams's *navigable*[p1061]*, or hand it off to external software*[p1073]*. Both happen in the steps below.*

3. 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 return the result of creating a document for inline content that doesn't have a DOM[p1112] given *navigationParams*'s navigable[p1061], *navigationParams*'s id[p1061], *navigationParams*'s navigation timing type[p1062], and *navigationParams*'s user involvement[p1062].

4. Otherwise, the document's *type* is such that the resource will not affect *navigationParams*'s navigable[p1061], e.g., because the resource is to be handed to an external application or because it is an unknown type that will be processed by handle as a download[p323]. Hand-off to external software[p1073] given *navigationParams*'s response[p1061], *navigationParams*'s navigable[p1061], *navigationParams*'s final sandboxing flag set[p1061], *sourceSnapshotParams*'s has transient activation[p1060], and *initiatorOrigin*.

5. Return null.

7.4.6 Applying the history step §[p1089]

For both navigation and traversal, once we have an idea of where we want to head to in the session history, much of the work comes about in applying that notion to the traversable navigable[p1037] and the relevant Document[p135]. For navigations, this work generally occurs toward the end of the process; for traversals, it is the beginning.

7.4.6.1 Updating the traversable §p1090

Ensuring a traversable[p1037] ends up at the right session history step is particularly complex, as it can involve coordinating across multiple navigable[p1035] descendants of the traversable, populating[p1079] them in parallel, and then synchronizing back up to ensure everyone has the same view of the result. This is further complicated by the existence of synchronous same-document navigations being mixed together with cross-document navigations, and how web pages have come to have certain relative timing expectations.

A **changing navigable continuation state** is used to store information during the apply the history step[p1091] algorithm, allowing parts of the algorithm to continue only after other parts have finished. It is a struct with:

displayed document

A `Document`[p135]

target entry

A session history entry[p1053]

navigable

A navigable[p1035]

update only

A boolean

Although all updates to the traversable navigable[p1037] end up in the same apply the history step[p1091] algorithm, each possible entry point comes along with some minor customizations:

To **update for navigable creation/destruction** given a traversable navigable[p1037] *traversable*:

1. Let *step* be *traversable*'s current session history step[p1037].
2. Return the result of applying the history step[p1091] *step* to *traversable* given false, null, null, "`none`[p1062]", and null.

To **apply the push/replace history step** given a non-negative integer *step* to a traversable navigable[p1037] *traversable*, given a history handling behavior[p1062] *historyHandling* and a user navigation involvement[p1062] *userInvolvement*:

1. Return the result of applying the history step[p1091] *step* to *traversable* given false, null, null, *userInvolvement*, and *historyHandling*.

Note

Apply the push/replace history step[p1090] never passes source snapshot params[p1060] or an initiator navigable[p1035] to apply the history step[p1091]. This is because those checks are done earlier in the navigation[p1063] algorithm.

To **apply the reload history step** to a traversable navigable[p1037] *traversable* given user navigation involvement[p1062] *userInvolvement*:

1. Let *step* be *traversable*'s current session history step[p1037].
2. Return the result of applying the history step[p1091] *step* to *traversable* given true, null, null, *userInvolvement*, and "`reload`[p997]".

Note

Apply the reload history step[p1090] never passes source snapshot params[p1060] or an initiator navigable[p1035] to apply the history step[p1091]. This is because reloading is always treated as if it were done by the navigable[p1035] itself, even in cases like `parent.location.reload()`.

To **apply the traverse history step** given a non-negative integer *step* to a traversable navigable[p1037] *traversable*, with source snapshot params[p1060] *sourceSnapshotParams*, navigable[p1035] *initiatorToCheck*, and user navigation involvement[p1062] *userInvolvement*:

1. Return the result of applying the history step[p1091] *step* to *traversable* given true, *sourceSnapshotParams*, *initiatorToCheck*, *userInvolvement*, and "`traverse`[p997]".

To **resume applying the traverse history step** given a non-negative integer *step*, a traversable navigable[p1037] *traversable*, and user navigation involvement[p1062] *userInvolvement*, apply[p1091] *step* to *traversable* given false, null, null, *userInvolvement*, and "`traverse`[p997]".

Note

When resuming a traverse, we are already past the cancelation, initiator, and source snapshot checks, and this traversal has already been determined to be a same-document traversal. Hence, we can pass false and null for those arguments.

Now for the algorithm itself.

To **apply the history step** given a non-negative integer *step* to a traversable navigable[p1037] *traversable*, with boolean *checkForCancelation*, source snapshot params[p1060]-or-null *sourceSnapshotParams*, navigable[p1035]-or-null *initiatorToCheck*, user navigation involvement[p1062] *userInvolvement*, and NavigationType[p996]-or-null *navigationType*, perform the following steps. They return "`initiator-disallowed`", "`canceled-by-beforeunload`", "`canceled-by-navigate`", or "`applied`".

1. Assert: This is running within *traversable*'s session history traversal queue[p1037].
2. Let *targetStep* be the result of getting the used step[p1097] given *traversable* and *step*.
3. If *initiatorToCheck* is not null:
    1. Assert: *sourceSnapshotParams* is not null.
    2. For each *navigable* of get all navigables whose current session history entry will change or reload[p1098]: if *initiatorToCheck* is not allowed by sandboxing to navigate[p1074] *navigable* given *sourceSnapshotParams*, then return "`initiator-disallowed`".
4. Let *navigablesCrossingDocuments* be the result of getting all navigables that might experience a cross-document traversal[p1099] given *traversable* and *targetStep*.
5. If *checkForCancelation* is true, and the result of checking if unloading is canceled[p1074] given *navigablesCrossingDocuments*, *traversable*, *targetStep*, and *userInvolvement* is not "`continue`", then return that result.
6. Let *changingNavigables* be the result of get all navigables whose current session history entry will change or reload[p1098] given *traversable* and *targetStep*.
7. Let *nonchangingNavigablesThatStillNeedUpdates* be the result of getting all navigables that only need history object length/index update[p1098] given *traversable* and *targetStep*.
8. For each *navigable* of *changingNavigables*:
    1. Let *targetEntry* be the result of getting the target history entry[p1098] given *navigable* and *targetStep*.
    2. Set *navigable*'s current session history entry[p1035] to *targetEntry*.
    3. If *targetEntry*'s document[p1053] is not *navigable*'s active document[p1036], then queue a global task[p1195] on the navigation and traversal task source[p1204] of *navigable*'s active window[p1036] to run these steps:
        1. Let *navigation* be *navigable*'s active window[p1036]'s navigation API[p995].
        2. Set *navigation*'s ongoing navigate event[p1007] to null.
        3. Set *navigation*'s ongoing API method tracker[p1007] to null.

        Note

        This prevents the navigateerror[p1575] event and the corresponding signal[p1016] from firing as a result of a successful cross-document navigation.

    4. Set *navigable*'s ongoing navigation[p1076] to "`traversal`".
9. Let *totalChangeJobs* be the size of *changingNavigables*.
10. Let *completedChangeJobs* be 0.
11. Let *changingNavigableContinuations* be an empty queue of changing navigable continuation states[p1090].

    Note

    This queue is used to split the operations on changingNavigables *into two parts. Specifically,* changingNavigableContinuations *holds data for the second part[p1094].*

12. For each *navigable* of *changingNavigables*, queue a global task[p1195] on the navigation and traversal task source[p1204] of

navigable's active window[p1036] to run the steps:

Note

This set of steps are split into two parts to allow synchronous navigations to be processed before documents unload. State is stored in changingNavigableContinuations *for the second part[p1094].*

1. Let *displayedEntry* be *navigable*'s active session history entry[p1035].
2. Let *targetEntry* be *navigable*'s current session history entry[p1035].
3. Let *changingNavigableContinuation* be a changing navigable continuation state[p1090] with:

   displayed document[p1090]
   : *displayedEntry*'s document[p1053]

   target entry[p1090]
   : *targetEntry*

   navigable[p1090]
   : *navigable*

   update-only[p1090]
   : false

4. If *displayedEntry* is *targetEntry* and *targetEntry*'s document state[p1053]'s reload pending[p1055] is false:

   1. Set *changingNavigableContinuation*'s update-only[p1090] to true.
   2. Enqueue *changingNavigableContinuation* on *changingNavigableContinuations*.
   3. Abort these steps.

   Note

   This case occurs due to a synchronous navigation[p1072] which already updated the active session history entry[p1035].

5. Switch on *navigationType*:

   ↪ "`reload`[p997]"
   : Assert: *targetEntry*'s document state[p1053]'s reload pending[p1055] is true.

   ↪ "`traverse`[p997]"
   : Assert: *targetEntry*'s document state[p1053]'s ever populated[p1055] is true.

   ↪ "`replace`[p996]"
   : Assert: *targetEntry*'s step[p1053] is *displayedEntry*'s step[p1053] and *targetEntry*'s document state[p1053]'s ever populated[p1055] is false.

   ↪ "`push`[p996]"
   : Assert: *targetEntry*'s step[p1053] is *displayedEntry*'s step[p1053] + 1 and *targetEntry*'s document state[p1053]'s ever populated[p1055] is false.

6. Let *oldOrigin* be *targetEntry*'s document state[p1053]'s origin[p1054].
7. If all of the following are true:

   - *navigationType* is "`traverse`[p997]";
   - *navigable* is not *traversable*;
   - *targetEntry* is not *displayedEntry*; and
   - *oldOrigin* is the same[p940] as *displayedEntry*'s document state[p1053]'s origin[p1054],

   then:

   1. Let *navigation* be *navigable*'s active window[p1036]'s navigation API[p995].
   2. Fire a traverse `navigate` event[p1019] at *navigation* given *targetEntry* and *userInvolvement*.

8. If *targetEntry*'s document[p1053] is null, or *targetEntry*'s document state[p1053]'s reload pending[p1055] is true:

    1. Let *navTimingType* be "back_forward" if *targetEntry*'s document[p1053] is null; otherwise "reload".
    2. Let *targetSnapshotParams* be the result of snapshotting target snapshot params[p1061] given *navigable*.
    3. Let *potentiallyTargetSpecificSourceSnapshotParams* be *sourceSnapshotParams*.
    4. If *potentiallyTargetSpecificSourceSnapshotParams* is null, then set it to the result of snapshotting source snapshot params[p1060] given *navigable*'s active document[p1036].

        Note

        In this case there is no clear source of the traversal/reload. We treat this situation as if navigable *navigated itself, but note that some properties of* targetEntry*'s original initiator are preserved in* targetEntry*'s document state[p1053], such as the initiator origin[p1054] and referrer[p1054], which will appropriately influence the navigation.*

    5. Set *targetEntry*'s document state[p1053]'s reload pending[p1055] to false.
    6. Let *allowPOST* be *targetEntry*'s document state[p1053]'s reload pending[p1055].
    7. In parallel[p44], attempt to populate the history entry's document[p1079] for *targetEntry*, given *navigable*, *potentiallyTargetSpecificSourceSnapshotParams*, *targetSnapshotParams*, *userInvolvement*, with *allowPOST*[p1079] set to *allowPOST* and *completionSteps*[p1079] set to queue a global task[p1195] on the navigation and traversal task source[p1204] given *navigable*'s active window[p1036] to run *afterDocumentPopulated*.

    Otherwise, run *afterDocumentPopulated* immediately[p44].

    In both cases, let *afterDocumentPopulated* be the following steps:

    1. If *targetEntry*'s document[p1053] is null, then set *changingNavigableContinuation*'s update-only[p1090] to true.

        Note

        This means we tried to populate the document, but were unable to do so, e.g. because of the server returning a 204.

        Note

        These kinds of failed navigations or traversals will not be signaled to the navigation API[p992] (e.g., through the promises of any navigation API method tracker[p1007], or the navigateerror[p1575] *event). Doing so would leak information about the timing of responses from other origins, in the cross-origin case, and providing different results in the cross-origin vs. same-origin cases was deemed too confusing.*

        However, implementations could use this opportunity to clear any promise handlers for the navigation.transition.finished[p1012] *promise, as they are guaranteed at this point to never run. And, they might wish to report a warning to the console if any part of the navigation API initiated these navigations, to make it clear to the web developer why their promises will never settle and events will never fire.*

    2. If *targetEntry*'s document[p1053]'s origin is not same origin[p940] with *oldOrigin*, then set *targetEntry*'s classic history API state[p1053] to StructuredSerializeForStorage[p128](null).

        Note

        This clears history state when the origin changed vs a previous load of targetEntry *without a redirect occuring. This can happen due to a change in CSP sandbox headers.*

    3. If all of the following are true:

        - *navigable*'s parent[p1035] is null;
        - *targetEntry*'s document[p1053]'s browsing context[p1046] is not an auxiliary browsing context[p1046] whose opener browsing context[p1046] is non-null; and
        - *targetEntry*'s document[p1053]'s origin is not same origin[p940] with *oldOrigin*,

then set *targetEntry*'s document state[p1053]'s navigable target name[p1055] to the empty string.

4. Enqueue *changingNavigableContinuation* on *changingNavigableContinuations*.

Note

The rest of this job runs later[p1094] in this algorithm.

13. Let *navigablesThatMustWaitBeforeHandlingSyncNavigation* be an empty set.

14. While *completedChangeJobs* does not equal *totalChangeJobs*:

    1. If *traversable*'s running nested apply history step[p1037] is false:

        1. While *traversable*'s session history traversal queue[p1037]'s algorithm set[p1056] contains one or more synchronous navigation steps[p1056] with a target navigable[p1056] not contained in *navigablesThatMustWaitBeforeHandlingSyncNavigation*:

            1. Let *steps* be the first item in *traversable*'s session history traversal queue[p1037]'s algorithm set[p1056] that is synchronous navigation steps[p1056] with a target navigable[p1056] not contained in *navigablesThatMustWaitBeforeHandlingSyncNavigation*.

            2. Remove *steps* from *traversable*'s session history traversal queue[p1037]'s algorithm set[p1056].

            3. Set *traversable*'s running nested apply history step[p1037] to true.

            4. Run *steps*.

            5. Set *traversable*'s running nested apply history step[p1037] to false.

        Note

        Synchronous navigations that are intended to take place before this traversal jump the queue at this point, so they can be added to the correct place in traversable's *session history entries[p1037] before this traversal potentially unloads their document. More details can be found here[p1056].*

    2. Let *changingNavigableContinuation* be the result of dequeuing from *changingNavigableContinuations*.

    3. If *changingNavigableContinuation* is nothing, then continue.

    4. Let *displayedDocument* be *changingNavigableContinuation*'s displayed document[p1090].

    5. Let *targetEntry* be *changingNavigableContinuation*'s target entry[p1090].

    6. Let *navigable* be *changingNavigableContinuation*'s navigable[p1090].

    7. Let (*scriptHistoryLength*, *scriptHistoryIndex*) be the result of getting the history object length and index[p1097] given *traversable* and *targetStep*.

    Note

    These values might have changed since they were last calculated.

    8. Append *navigable* to *navigablesThatMustWaitBeforeHandlingSyncNavigation*.

    Note

    Once a navigable has reached this point in traversal, additionally queued synchronous navigation steps are likely to be intended to occur after this traversal rather than before it, so they no longer jump the queue. More details can be found here[p1056].

    9. Let *entriesForNavigationAPI* be the result of getting session history entries for the navigation API[p1058] given *navigable* and *targetStep*.

    10. If *changingNavigableContinuation*'s update-only[p1090] is true, or *targetEntry*'s document[p1053] is *displayedDocument*:

    Note

    This is a same-document navigation: we proceed without unloading.

        1. Set *navigable*'s ongoing navigation[p1076] to null.

Note

This allows new navigations[p1063] of navigable *to start, whereas during the traversal they were blocked.*

2. Queue a global task[p1195] on the navigation and traversal task source[p1204] given *navigable*'s active window[p1036] to perform *afterPotentialUnloads*.

11. Otherwise:

    1. Assert: *navigationType* is not null.

    2. Deactivate[p1095] *displayedDocument*, given *userInvolvement*, *targetEntry*, *navigationType*, and *afterPotentialUnloads*.

12. In both cases, let *afterPotentialUnloads* be the following steps:

    1. Let *previousEntry* be *navigable*'s active session history entry[p1035].

    2. If *changingNavigableContinuation*'s update-only[p1090] is false, then activate history entry[p1097] *targetEntry* for *navigable*.

    3. Let *updateDocument* be an algorithm step which performs update document for history step application[p1100] given *targetEntry*'s document[p1053], *targetEntry*, *changingNavigableContinuation*'s update-only[p1090], *scriptHistoryLength*, *scriptHistoryIndex*, *navigationType*, *entriesForNavigationAPI*, and *previousEntry*.

    4. If *targetEntry*'s document[p1053] is equal to *displayedDocument*, then perform *updateDocument*.

    5. Otherwise, queue a global task[p1195] on the navigation and traversal task source[p1204] given *targetEntry*'s document[p1053]'s relevant global object[p1151] to perform *updateDocument*.

    6. Increment *completedChangeJobs*.

15. Let *totalNonchangingJobs* be the size of *nonchangingNavigablesThatStillNeedUpdates*.

Note

This step onwards deliberately waits for all the previous operations to complete, as they include processing synchronous navigations[p1094] which will also post tasks to update history length and index.

16. Let *completedNonchangingJobs* be 0.

17. Let (*scriptHistoryLength*, *scriptHistoryIndex*) be the result of getting the history object length and index[p1097] given *traversable* and *targetStep*.

18. For each *navigable* of *nonchangingNavigablesThatStillNeedUpdates*, queue a global task[p1195] on the navigation and traversal task source[p1204] given *navigable*'s active window[p1036] to run the steps:

    1. Let *document* be *navigable*'s active document[p1036].

    2. Set *document*'s history object[p988]'s index[p988] to *scriptHistoryIndex*.

    3. Set *document*'s history object[p988]'s length[p988] to *scriptHistoryLength*.

    4. Increment *completedNonchangingJobs*.

19. Wait for *completedNonchangingJobs* to equal *totalNonchangingJobs*.

20. Set *traversable*'s current session history step[p1037] to *targetStep*.

21. Return "`applied`".

To **deactivate a document for a cross-document navigation** given a `Document`[p135] *displayedDocument*, a user navigation involvement[p1062] *userNavigationInvolvement*, a session history entry[p1053] *targetEntry*, a `NavigationType`[p996] *navigationType*, and *afterPotentialUnloads*, which is an algorithm that receives no arguments:

1. Let *navigable* be *displayedDocument*'s node navigable[p1036].

2. Let *potentiallyTriggerViewTransition* be false.

3. Let *isBrowserUINavigation* be true if *userNavigationInvolvement* is "browser UI"; otherwise false.
4. Set *potentiallyTriggerViewTransition* to the result of calling can navigation trigger a cross-document view-transition? given *displayedDocument*, *targetEntry*'s document, *navigationType*, and *isBrowserUINavigation*.
5. If *potentiallyTriggerViewTransition* is false:
    1. Let *firePageSwapBeforeUnload* be the following step:
        1. Fire the pageswap event given *displayedDocument*, *targetEntry*, *navigationType*, and null.
    2. Set *navigable*'s ongoing navigation to null.

    Note

    This allows new navigations of navigable *to start, whereas during the traversal they were blocked.*

    3. Unload a document and its descendants given *displayedDocument*, *targetEntry*'s document, *afterPotentialUnloads*, and *firePageSwapBeforeUnload*.
6. Otherwise, queue a global task on the navigation and traversal task source given *navigable*'s active window to run the steps:
    1. Let *proceedWithNavigationAfterViewTransitionCapture* be the following step:
        1. Append the following session history traversal steps to *navigable*'s traversable navigable:
            1. Set *navigable*'s ongoing navigation to null.

            Note

            This allows new navigations of navigable *to start, whereas during the traversal they were blocked.*

            2. Unload a document and its descendants given *displayedDocument*, *targetEntry*'s document, and *afterPotentialUnloads*.
    2. Let *viewTransition* be the result of setting up a cross-document view-transition given *displayedDocument*, *targetEntry*'s document, *navigationType*, and *proceedWithNavigationAfterViewTransitionCapture*.
    3. Fire the pageswap event given *displayedDocument*, *targetEntry*, *navigationType*, and *viewTransition*.
    4. If *viewTransition* is null, then run *proceedWithNavigationAfterViewTransitionCapture*.

    Note

    In the case where a view transition started, the view transitions algorithms are responsible for calling proceedWithNavigationAfterViewTransitionCapture.

To **fire the pageswap event** given a Document *displayedDocument*, a session history entry *targetEntry*, a NavigationType *navigationType*, and a ViewTransition-or-null *viewTransition*:

1. Assert: this is running as part of a task queued on *displayedDocument*'s relevant agent's event loop.
2. Let *navigation* be *displayedDocument*'s relevant global object's navigation API.
3. Let *activation* be null.
4. If all of the following are true:
    - *targetEntry*'s document's origin is same origin with *displayedDocument*'s origin; and
    - *targetEntry*'s document's was created via cross-origin redirects is false, or *targetEntry*'s document's latest entry is not null,

    then:
    1. Let *destinationEntry* be determined by switching on *navigationType*:

        ↪ "**reload**"

        The current entry of *navigation*

↪ "**traverse**[p997]"

The NavigationHistoryEntry[p999] in *navigation*'s entry list[p996] whose session history entry[p1000] is *targetEntry*

↪ "**push**[p996]"

↪ "**replace**[p996]"

A new NavigationHistoryEntry[p999] in *displayedDocument*'s relevant realm[p1151] with its session history entry[p1000] set to *targetEntry*

2. Set *activation* to a new NavigationActivation[p1012] created in *displayedDocument*'s relevant realm[p1151], with

old entry[p1013]
: the current entry[p996] of *navigation*

new entry[p1013]
: *destinationEntry*

navigation type[p1013]
: *navigationType*

Note

This means that a cross-origin redirect during a navigation would result in a null activation[p1028] in the old document's PageSwapEvent[p1028], unless the new document is being restored from bfcache[p1054].

5. Fire an event named pageswap[p1576] at *displayedDocument*'s relevant global object[p1151], using PageSwapEvent[p1028] with its activation[p1028] set to *activation*, and its viewTransition[p1028] set to *viewTransition*.

To **activate history entry** session history entry[p1053] *entry* for navigable[p1035] *navigable*:

1. Save persisted state[p1105] to the navigable[p1035]'s active session history entry[p1035].
2. Let *newDocument* be *entry*'s document[p1053].
3. Assert: *newDocument*'s is initial about:blank[p136] is false, i.e., we never traverse back to the initial about:blank[p136] Document[p135] because it always gets replaced[p1064] when we navigate away from it.
4. Set *navigable*'s active session history entry[p1035] to *entry*.
5. Make active[p1101] *newDocument*.
6. Set the initial visibility state[p865] of *newDocument* to *navigable*'s traversable navigable[p1037]'s system visibility state[p1037].

To **get the used step** given a traversable navigable[p1037] *traversable*, and a non-negative integer *step*, perform the following steps. They return a non-negative integer.

1. Let *steps* be the result of getting all used history steps[p1059] within *traversable*.
2. Return the greatest item in *steps* that is less than or equal to *step*.

Note

This caters for situations where there's no session history entry[p1053] with step[p1053] step, due to the removal of a navigable[p1035].

To **get the history object length and index** given a traversable navigable[p1037] *traversable*, and a non-negative integer *step*, perform the following steps. They return a tuple of two non-negative integers.

1. Let *steps* be the result of getting all used history steps[p1059] within *traversable*.
2. Let *scriptHistoryLength* be the size of *steps*.
3. Assert: *steps* contains *step*.

Note

It is assumed that step has been adjusted by getting the used step[p1097].

4. Let *scriptHistoryIndex* be the index of *step* in *steps*.
5. Return (*scriptHistoryLength*, *scriptHistoryIndex*).

To **get all navigables whose current session history entry will change or reload** given a traversable navigable[p1037] *traversable*, and a non-negative integer *targetStep*, perform the following steps. They return a list of navigables[p1035].

1. Let *results* be an empty list.
2. Let *navigablesToCheck* be « *traversable* ».

   Note

   This list is extended in the loop below.

3. For each *navigable* of *navigablesToCheck*:
   1. Let *targetEntry* be the result of getting the target history entry[p1098] given *navigable* and *targetStep*.
   2. If *targetEntry* is not *navigable*'s current session history entry[p1035] or *targetEntry*'s document state[p1053]'s reload pending[p1055] is true, then append *navigable* to *results*.
   3. If *targetEntry*'s document[p1053] is *navigable*'s active document[p1036], and *targetEntry*'s document state[p1053]'s reload pending[p1055] is false, then extend *navigablesToCheck* with the child navigables[p1039] of *navigable*.

      Note

      Adding child navigables[p1039] to navigablesToCheck *means those navigables will also be checked by this loop. Child navigables[p1039] are only checked if the* navigable's *active document[p1036] will not change as part of this traversal.*

4. Return *results*.

To **get all navigables that only need history object length/index update** given a traversable navigable[p1037] *traversable*, and a non-negative integer *targetStep*, perform the following steps. They return a list of navigables[p1035].

Note

Other navigables[p1035] might not be impacted by the traversal. For example, if the response is a 204, the currently active document will remain. Additionally, going 'back' after a 204 will change the current session history entry[p1035], but the active session history entry[p1035] will already be correct.

1. Let *results* be an empty list.
2. Let *navigablesToCheck* be « *traversable* ».

   Note

   This list is extended in the loop below.

3. For each *navigable* of *navigablesToCheck*:
   1. Let *targetEntry* be the result of getting the target history entry[p1098] given *navigable* and *targetStep*.
   2. If *targetEntry* is *navigable*'s current session history entry[p1035] and *targetEntry*'s document state[p1053]'s reload pending[p1055] is false:
      1. Append *navigable* to *results*.
      2. Extend *navigablesToCheck* with *navigable*'s child navigables[p1039].

         Note

         Adding child navigables[p1039] to navigablesToCheck *means those navigables will also be checked by this loop. child navigables[p1039] are only checked if the* navigable's *active document[p1036] will not change as part of this traversal.*

4. Return *results*.

To **get the target history entry** given a navigable[p1035] *navigable*, and a non-negative integer *step*, perform the following steps. They return a session history entry[p1053].

1. Let *entries* be the result of getting session history entries[p1058] for *navigable*.

2. Return the item in *entries* that has the greatest step[p1053] less than or equal to *step*.

¶[p1099] **Example**

To see why getting the target history entry[p1098] returns the entry with the greatest step[p1053] less than or equal to the input step, consider the following Jake diagram[p1040]:

	0	1	2	3
top	/t			/t#foo
frames[0]	/i-0-a	/i-0-b		

For the input step 1, the target history entry for the `top` navigable is the `/t` entry, whose step[p1053] is 0, while the target history entry for the `frames[0]` navigable is the `/i-0-b` entry, whose step[p1053] is 1:

	0	***1***	2	3
top	***/t***			/t#foo
frames[0]	/i-0-a	***/i-0-b***		

Similarly, given the input step 3 we get the `top` entry whose step[p1053] is 3, and the `frames[0]` entry whose step[p1053] is 1:

	0	1	2	***3***
top	/t			***/t#foo***
frames[0]	/i-0-a	***/i-0-b***		

To **get all navigables that might experience a cross-document traversal** given a traversable navigable[p1037] *traversable*, and a non-negative integer *targetStep*, perform the following steps. They return a list of navigables[p1035].

Note

From traversable's *session history traversal queue*[p1037]*'s perspective, these documents are candidates for going cross-document during the traversal described by* targetStep*. They will not experience a cross-document traversal if the status code for their target document is HTTP 204 No Content.*

Note that if a given navigable[p1035] *might experience a cross-document traversal, this algorithm will return navigable*[p1035] *but not its child navigables*[p1039]*. Those would end up unloaded*[p1114]*, not traversed.*

1. Let *results* be an empty list.
2. Let *navigablesToCheck* be « *traversable* ».

   Note

   This list is extended in the loop below.

3. For each *navigable* of *navigablesToCheck*:

   1. Let *targetEntry* be the result of getting the target history entry[p1098] given *navigable* and *targetStep*.
   2. If *targetEntry*'s document[p1053] is not *navigable*'s active document[p1036] or *targetEntry*'s document state[p1053]'s reload pending[p1055] is true, then append *navigable* to *results*.

      Note

      Although navigable's *active history entry*[p1035] *can change synchronously, the new entry will always have the same* `Document`[p135]*, so accessing* navigable's *active document*[p1036] *is reliable.*

   3. Otherwise, extend *navigablesToCheck* with *navigable*'s child navigables[p1039].

      Note

      Adding child navigables[p1039] *to* navigablesToCheck *means those navigables will also be checked by this loop. Child navigables*[p1039] *are only checked if the* navigable's *active document*[p1036] *will not change as part of this traversal.*

4. Return *results*.

7.4.6.2 Updating the document §[p1100]

To **update document for history step application** given a Document[p135] *document*, a session history entry[p1053] *entry*, a boolean *doNotReactivate*, integers *scriptHistoryLength* and *scriptHistoryIndex*, NavigationType[p996]-or-null *navigationType*, an optional list of session history entries[p1053] *entriesForNavigationAPI*, and an optional session history entry[p1053] *previousEntryForActivation*:

1. Let *documentIsNew* be true if *document*'s latest entry[p1055] is null; otherwise false.
2. Let *documentsEntryChanged* be true if *document*'s latest entry[p1055] is not *entry*; otherwise false.
3. Set *document*'s history object[p988]'s index[p988] to *scriptHistoryIndex*.
4. Set *document*'s history object[p988]'s length[p988] to *scriptHistoryLength*.
5. Let *navigation* be *history*'s relevant global object[p1151]'s navigation API[p995].
6. If *documentsEntryChanged* is true:
    1. Let *oldURL* be *document*'s latest entry[p1055]'s URL[p1053].
    2. Set *document*'s latest entry[p1055] to *entry*.
    3. Restore the history object state[p1101] given *document* and *entry*.
    4. If *documentIsNew* is false:
        1. Assert: *navigationType* is not null.
        2. Update the navigation API entries for a same-document navigation[p998] given *navigation*, *entry*, and *navigationType*.
        3. Fire an event named popstate[p1576] at *document*'s relevant global object[p1151], using PopStateEvent[p1027], with the state[p1027] attribute initialized to *document*'s history object[p988]'s state[p988] and hasUAVisualTransition[p1027] initialized to true if a visual transition, to display a cached rendered state of the latest entry[p1055], was done by the user agent.
        4. Restore persisted state[p1105] given *entry*.
        5. If *oldURL*'s fragment is not equal to *entry*'s URL[p1053]'s fragment, then queue a global task[p1195] on the DOM manipulation task source[p1203] given *document*'s relevant global object[p1151] to fire an event named hashchange[p1575] at *document*'s relevant global object[p1151], using HashChangeEvent[p1027], with the oldURL[p1028] attribute initialized to the serialization of *oldURL* and the newURL[p1028] attribute initialized to the serialization of *entry*'s URL[p1053].
    5. Otherwise:
        1. Assert: *entriesForNavigationAPI* is given.
        2. Restore persisted state[p1105] given *entry*.
        3. Initialize the navigation API entries for a new document[p997] given *navigation*, *entriesForNavigationAPI*, and *entry*.
7. If all the following are true:
    - *previousEntryForActivation* is given;
    - *navigationType* is non-null; and
    - *navigationType* is "reload[p997]" or *previousEntryForActivation*'s document[p1053] is not *document*,

    then:
    1. If *navigation*'s activation[p1013] is null, then set *navigation*'s activation[p1013] to a new NavigationActivation[p1012] object in *navigation*'s relevant realm[p1151].
    2. Let *previousEntryIndex* be the result of getting the navigation API entry index[p996] of *previousEntryForActivation* within *navigation*.
    3. If *previousEntryIndex* is non-negative, then set *activation*'s old entry[p1013] to *navigation*'s entry list[p996][*previousEntryIndex*].

4. Otherwise, if all the following are true:

    - *navigationType* is "replace[p996]";
    - *previousEntryForActivation*'s document state[p1053]'s origin[p1054] is same origin[p940] with *document*'s origin; and
    - *previousEntryForActivation*'s document[p1053]'s is initial about:blank[p136] is false,

    then set *activation*'s old entry[p1013] to a new NavigationHistoryEntry[p999] in *navigation*'s relevant realm[p1151], whose session history entry[p1000] is *previousEntryForActivation*.

5. Set *activation*'s new entry[p1013] to *navigation*'s current entry[p996].
6. Set *activation*'s navigation type[p1013] to *navigationType*.

8. If *documentIsNew* is true:

    1. Assert: *document*'s during-loading navigation ID for WebDriver BiDi[p136] is not null.
    2. Invoke WebDriver BiDi navigation committed with *navigable* and a new WebDriver BiDi navigation status whose id is *document*'s during-loading navigation ID for WebDriver BiDi[p136], status is "committed", and url is *document*'s URL.
    3. Try to scroll to the fragment[p1102] for *document*.
    4. At this point **scripts may run for the newly-created document** *document*.

9. Otherwise, if *documentsEntryChanged* is false and *doNotReactivate* is false:

    1. Assert: *entriesForNavigationAPI* is given.
    2. Reactivate[p1101] *document* given *entry* and *entriesForNavigationAPI*.

Note

documentsEntryChanged *can be false for one of two reasons: either we are restoring from bfcache[p1054], or we are asynchronously finishing up a synchronous navigation which already synchronously set* document's *latest entry[p1055]. The* doNotReactivate *argument distinguishes between these two cases.*

To **restore the history object state** given Document[p135] *document* and session history entry[p1053] *entry*:

1. Let *targetRealm* be *document*'s relevant realm[p1151].
2. Let *state* be StructuredDeserialize[p128](*entry*'s classic history API state[p1053], *targetRealm*). If this throws an exception, catch it and let *state* be null.
3. Set *document*'s history object[p988]'s state[p988] to *state*.

To **make active** a Document[p135] *document*:

1. Let *window* be *document*'s relevant global object[p1151].
2. Set *document*'s browsing context[p1046]'s active document[p1046] to *document*.
3. Set *document*'s browsing context[p1046]'s WindowProxy[p977]'s [[Window]][p977] internal slot value to *window*.
4. Set *window*'s relevant settings object[p1151]'s execution ready flag[p1144].

To **reactivate** a Document[p135] *document* given a session history entry[p1053] *reactivatedEntry* and a list of session history entries[p1053] *entriesForNavigationAPI*:

Note

This algorithm updates document *after it has come out of bfcache[p1054], i.e., after it has been made fully active[p1051] again. Other specifications that want to watch for this change to the fully active[p1051] state are encouraged to add steps into this algorithm, so that the ordering of events that happen in effect of the change is clear.*

1. For each *formControl* of form controls in *document* with an autofill field name[p639] of "off[p635]", invoke the reset algorithm[p666] for *formControl*.

2. If *document*'s suspended timer handles[p1114] is not empty:

    1. Assert: *document*'s suspension time[p1114] is not zero.

    2. Let *suspendDuration* be the current high resolution time minus *document*'s suspension time[p1114].

    3. Let *activeTimers* be *document*'s relevant global object[p1151]'s map of active timers[p1256].

    4. For each *handle* in *document*'s suspended timer handles[p1114], if *activeTimers*[*handle*] exists, then increase *activeTimers*[*handle*] by *suspendDuration*.

3. Update the navigation API entries for reactivation[p997] given *document*'s relevant global object[p1151]'s navigation API[p995], *entriesForNavigationAPI*, and *reactivatedEntry*.

4. If *document*'s current document readiness[p141] is "`complete`", and *document*'s page showing[p1114] is false:

    1. Set *document*'s page showing[p1114] to true.

    2. Set *document*'s has been revealed[p1103] to false.

    3. Update the visibility state[p865] of *document* to "`visible`".

    4. Fire a page transition event[p1029] named `pageshow`[p1576] at *document*'s relevant global object[p1151] with true.

To **try to scroll to the fragment** for a `Document`[p135] *document*, perform the following steps in parallel[p44]:

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[p1195] on the navigation and traversal task source[p1204] given *document*'s relevant global object[p1151] to run these steps:

    1. If *document* has no parser, or its parser has stopped parsing[p1458], 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[p1104] given *document*.

    3. If *document*'s indicated part[p1104] is still null, then try to scroll to the fragment[p1102] for *document*.

To **make document unsalvageable**, given a `Document`[p135] *document* and a string *reason*:

1. Let *details* be a new not restored reason details[p1031] whose reason[p1031] is *reason*.

2. Append *details* to *document*'s bfcache blocking details[p137].

3. Set *document*'s *salvageable*[p1114] state to false.

To **build not restored reasons for document state** given `Document`[p135] *document*:

1. Let *notRestoredReasonsForDocument* be a new not restored reasons[p1035].

2. Set *notRestoredReasonsForDocument*'s URL[p1035] to *document*'s URL.

3. Let *container* be *document*'s node navigable[p1036]'s container[p1038].

4. If *container* is an `iframe`[p404] element:

    1. Let *src* be the empty string.

    2. If *container* has a `src`[p405] attribute:

        1. Let *src* be the result of encoding-parsing-and-serializing a URL[p101] given *container*'s `src`[p405] attribute's value, relative to *container*'s node document.

        2. If *src* is failure, then set *src* to *container*'s `src`[p405] attribute's value.

    3. Set *notRestoredReasonsForDocument*'s src[p1035] to *src*.

    4. Set *notRestoredReasonsForDocument*'s id[p1035] to *container*'s `id`[p162] attribute's value, or the empty string if it has no such attribute.

5. Set *notRestoredReasonsForDocument*'s name[p1035] to *container*'s name[p409] attribute's value, or the empty string if it has no such attribute.

5. Set *notRestoredReasonsForDocument*'s reasons[p1035] to a clone of *document*'s bfcache blocking details[p137].

6. For each *navigable* of *document*'s document-tree child navigables[p1041]:

    1. Let *childDocument* be *navigable*'s active document[p1036].

    2. Build not restored reasons for document state[p1102] given *childDocument*.

    3. Append *childDocument*'s not restored reasons[p1035] to *notRestoredReasonsForDocument*'s children[p1035].

7. Set *document*'s node navigable[p1036]'s active session history entry[p1035]'s document state[p1053]'s not restored reasons[p1055] to *notRestoredReasonsForDocument*.

To **build not restored reasons for a top-level traversable and its descendants** given top-level traversable[p1037] *topLevelTraversable*:

1. Build not restored reasons for document state[p1102] given *topLevelTraversable*'s active document[p1036].

2. Let *crossOriginDescendants* be an empty list.

3. For each *childNavigable* of *topLevelTraversable*'s active document[p1036]'s descendant navigables[p1041]:

    1. If *childNavigable*'s active document[p1036]'s origin is not same origin[p940] with *topLevelTraversable*'s active document[p1036]'s origin, then append *childNavigable* to *crossOriginDescendants*.

4. Let *crossOriginDescendantsPreventsBfcache* be false.

5. For each *crossOriginNavigable* of *crossOriginDescendants*:

    1. Let *reasonsForCrossOriginChild* be *crossOriginNavigable*'s active document[p1036]'s document state[p1054]'s not restored reasons[p1055].

    2. If *reasonsForCrossOriginChild*'s reasons[p1035] is not empty, set *crossOriginDescendantsPreventsBfcache* to true.

    3. Set *reasonsForCrossOriginChild*'s URL[p1035] to null.

    4. Set *reasonsForCrossOriginChild*'s reasons[p1035] to null.

    5. Set *reasonsForCrossOriginChild*'s children[p1035] to null.

6. If *crossOriginDescendantsPreventsBfcache* is true, make document unsalvageable[p1102] given *topLevelTraversable*'s active document[p1036] and "masked[p1031]".

7.4.6.3 Revealing the document §[p1103]

A Document[p135] has a boolean **has been revealed**, initially false. It is used to ensure that the pagereveal[p1576] event is fired once for each activation of the Document[p135] (once when it's rendered initially, and once for each reactivation[p1101]).

To **reveal** a Document[p135] *document*:

1. If *document*'s has been revealed[p1103] is true, then return.

2. Set *document*'s has been revealed[p1103] to true.

3. Let *transition* be the result of resolving inbound cross-document view-transition for *document*.

4. Fire an event named pagereveal[p1576] at *document*'s relevant global object[p1151], using PageRevealEvent[p1029] with its viewTransition[p1029] set to *transition*.

5. If *transition* is not null:

    1. Prepare to run script[p1166] given *document*'s relevant settings object[p1151].

    2. Activate *transition*.

3. Clean up after running script[p1166] given *document*'s relevant settings object[p1151].

Note

Activating a view transition might resolve/reject promises, so by wrapping the activation with prepare/cleanup we ensure those promises are handled before the next rendering step.

Note

Though `pagereveal`[p1576] *is guaranteed to be fired during the first update the rendering*[p1198] *step that displays an up-to-date version of the page, user agents are free to display a cached frame of the page before firing it. This prevents the presence of a* `pagereveal`[p1576] *handler from delaying the presentation of such cached frame.*

7.4.6.4 Scrolling to a fragment §[p1104]

To **scroll to the fragment** given a `Document`[p135] *document*:

1. If *document*'s indicated part[p1104] is null, then set *document*'s target element[p1104] to null.
2. Otherwise, if *document*'s indicated part[p1104] is top of the document[p1104]:
    1. Set *document*'s target element[p1104] to null.
    2. Scroll to the beginning of the document for *document*. [CSSOMVIEW][p1581]
    3. Return.
3. Otherwise:
    1. Assert: *document*'s indicated part[p1104] is an element.
    2. Let *target* be *document*'s indicated part[p1104].
    3. Set *document*'s target element[p1104] to *target*.
    4. Run the ancestor revealing algorithm[p864] on *target*.
    5. Scroll *target* into view, with *behavior* set to "auto", *block* set to "start", and *inline* set to "nearest". [CSSOMVIEW][p1581]
    6. Run the focusing steps[p881] for *target*, with the `Document`[p135]'s viewport as the *fallback target*.
    7. Move the sequential focus navigation starting point[p884] to *target*.

A `Document`[p135]'s **indicated part** is the one that its URL's fragment identifies, or null if the fragment does not identify anything. 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`[p135] (for example, the processing of fragments for XML MIME types is the responsibility of RFC7303). [RFC7303][p1586]

There is also a **target element** for each `Document`[p135], which is used in defining the `:target`[p822] pseudo-class and is updated by the above algorithm. It is initially null.

For an HTML document *document*, its indicated part[p1104] is the result of selecting the indicated part[p1104] given *document* and *document*'s URL.

To **select the indicated part** given a `Document`[p135] *document* and a URL *url*:

1. If *document*'s URL does not equal *url* with *exclude fragments* set to true, then return null.
2. Let *fragment* be *url*'s fragment.
3. If *fragment* is the empty string, then return the special value **top of the document**.
4. Let *potentialIndicatedElement* be the result of finding a potential indicated element[p1105] given *document* and *fragment*.
5. If *potentialIndicatedElement* is not null, then return *potentialIndicatedElement*.
6. Let *fragmentBytes* be the result of percent-decoding *fragment*.

7. Let *decodedFragment* be the result of running UTF-8 decode without BOM on *fragmentBytes*.
8. Set *potentialIndicatedElement* to the result of finding a potential indicated element[p1105] given *document* and *decodedFragment*.
9. If *potentialIndicatedElement* is not null, then return *potentialIndicatedElement*.
10. If *decodedFragment* is an ASCII case-insensitive match for the string `top`, then return the top of the document[p1104].
11. Return null.

To **find a potential indicated element** given a `Document`[p135] *document* and a string *fragment*, run these steps:

1. If there is an element in the document tree whose root is *document* and that has an ID equal to *fragment*, then return the first such element in tree order.
2. If there is an `a`[p267] element in the document tree whose root is *document* that has a `name`[p1531] attribute whose value is equal to *fragment*, then return the first such element in tree order.
3. Return null.

7.4.6.5 Persisted history entry state §[p1105]

To **save persisted state** to a session history entry[p1053] *entry*:

1. Set the scroll position data[p1053] of *entry* to contain the scroll positions for all of *entry*'s document[p1053]'s restorable scrollable regions[p1106].
2. Optionally, update *entry*'s persisted user state[p1053] 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[p1053] *entry*:

1. If *entry*'s scroll restoration mode[p1053] is "`auto`[p1054]", and *entry*'s document[p1053]'s relevant global object[p1151]'s navigation API[p995]'s suppress normal scroll restoration during ongoing navigation[p1007] is false, then restore scroll position data[p1106] given *entry*.

   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.

   Note

   If suppress normal scroll restoration during ongoing navigation[p1007] is true, then restoring scroll position data[p1106] might still happen at a later point, as part of finishing[p1025] the relevant `NavigateEvent`[p1013]*, or via a* `navigateEvent.scroll()`[p1016] *method call.*

2. Optionally, update other aspects of *entry*'s document[p1053] and its rendering, for instance values of form fields, that the user agent had previously recorded in *entry*'s persisted user state[p1053].

   Note

   This can even include updating the `dir`[p167] *attribute of* `textarea`[p606] *elements or* `input`[p539] *elements whose* `type`[p541] *attribute is in the Text[p546], Search[p546], Telephone[p547], URL[p547], or Email[p548] state, if the persisted state includes the directionality of user input in such controls.*

   Note

   Restoring the value of form controls as part of this process does not fire any `input` *or* `change`[p1575] *events, but can trigger the* `formStateRestoreCallback` *of form-associated custom elements[p797].*

Each `Document`[p135] has a boolean **has been scrolled by the user**, initially false. If the user scrolls the document, the user agent must set that document's has been scrolled by the user[p1105] to true.

The **restorable scrollable regions** of a Document[p135] *document* are *document*'s viewport, and all of *document*'s scrollable regions excepting any navigable containers[p1038].

Note

Child navigable[p1039] scroll restoration is handled as part of state restoration for the session history entry[p1053] for those navigables[p1035]' Document[p135]s.

To **restore scroll position data** given a session history entry[p1053] *entry*:

1. Let *document* be *entry*'s document[p1053].
2. If *document*'s has been scrolled by the user[p1105] is true, then the user agent should return.
3. The user agent should attempt to use *entry*'s scroll position data[p1053] to restore the scroll positions of *entry*'s document[p1053]'s restorable scrollable regions[p1106]. The user agent may continue to attempt to do so periodically, until *document*'s has been scrolled by the user[p1105] becomes true.

Note

This is formulated as an attempt*, which is potentially repeated until success or until the user scrolls, due to the fact that relevant content indicated by the scroll position data[p1053] might take some time to load from the network.*

Note

Scroll restoration might be affected by scroll anchoring. [CSSSCROLLANCHORING][p1581]

7.5 Document lifecycle §[p1106]

7.5.1 Shared document creation infrastructure §[p1106]

When loading a document using one of the below algorithms, we use the following steps to **create and initialize a `Document` object**, given a type *type*, content type *contentType*, and navigation params[p1061] *navigationParams*:

Note

Document[p135] *objects are also created when creating a new browsing context and document[p1047]; such initial* `about:blank`[p136] Document[p135] *are never created by this algorithm. Also, browsing context[p1046]-less* Document[p135] *objects can be created via various APIs, such as* `document.implementation.createHTMLDocument()`.

1. Let *browsingContext* be the result of obtaining a browsing context to use for a navigation response[p949] given *navigationParams*.

   Note

   This can result in a browsing context group switch[p947], in which case browsingContext *will be a newly-created[p1047] browsing context[p1046] instead of being* navigationParams's *navigable[p1061]'s active browsing context[p1036]. In such a case, the created* Window[p965]*,* Document[p135]*, and agent will not end up being used; because the created* Document[p135]*'s origin is opaque[p939], we will end up creating a new agent and* Window[p965] *later in this algorithm[p1107] to go along with the new* Document[p135].

2. Let *permissionsPolicy* be the result of creating a permissions policy from a response given *navigationParams*'s navigable[p1061]'s container[p1038], *navigationParams*'s origin[p1061], and *navigationParams*'s response[p1061]. [PERMISSIONSPOLICY][p1585]

   Note

   The creating a permissions policy from a response algorithm makes use of the passed origin[p939]. If `document.domain`[p942] *has been used for* navigationParams's *navigable[p1061]'s container document[p1038], then its origin cannot be same origin-domain[p940] with the passed origin, because these steps run before the* document *is created, so it cannot itself yet have used* `document.domain`[p942]*. Note that this means that Permissions Policy checks are less permissive compared to doing a same origin[p940] check instead.*

   See below for some examples of this in action.

3. Let *creationURL* be *navigationParams*'s response[p1061]'s URL.

4. If *navigationParams*'s request[p1061] is non-null, then set *creationURL* to *navigationParams*'s request[p1061]'s current URL.

5. Let *window* be null.

6. If *browsingContext*'s active document[p1046]'s is initial about:blank[p136] is true, and *browsingContext*'s active document[p1046]'s origin is same origin-domain[p940] with *navigationParams*'s origin[p1061], then set *window* to *browsingContext*'s active window[p1046].

   Note

   This means that both the initial about:blank[p136] Document[p135], and the new Document[p135] that is about to be created, will share the same Window[p965] object.

7. Otherwise:

   1. Let *oacHeader* be the result of getting a structured field value given `Origin-Agent-Cluster[p944]` and "item" from *navigationParams*'s response[p1061]'s header list.

   2. Let *requestsOAC* be true if *oacHeader* is not null and *oacHeader*[0] is the boolean true; otherwise false.

   3. If *navigationParams*'s reserved environment[p1061] is a non-secure context[p1152], then set *requestsOAC* to false.

   4. Let *agent* be the result of obtaining a similar-origin window agent[p1141] given *navigationParams*'s origin[p1061], *browsingContext*'s group[p1050], and *requestsOAC*.

   5. Let *realmExecutionContext* be the result of creating a new realm[p1145] given *agent* and the following customizations:

      - For the global object, create a new Window[p965] object.
      - For the global **this** binding, use *browsingContext*'s WindowProxy[p977] object.

   6. Set *window* to the global object[p1145] of *realmExecutionContext*'s Realm component.

   7. Let *topLevelCreationURL* be *creationURL*.

   8. Let *topLevelOrigin* be *navigationParams*'s origin[p1061].

   9. If *navigable*'s container[p1038] is not null:

      1. Let *parentEnvironment* be *navigable*'s container[p1038]'s relevant settings object[p1151].

      2. Set *topLevelCreationURL* to *parentEnvironment*'s top-level creation URL[p1144].

      3. Set *topLevelOrigin* to *parentEnvironment*'s top-level origin[p1144].

   10. Set up a window environment settings object[p976] with *creationURL*, *realmExecutionContext*, *navigationParams*'s reserved environment[p1061], *topLevelCreationURL*, and *topLevelOrigin*.

   Note

   This is the usual case, where the new Document[p135] we're about to create gets a new Window[p965] to go along with it.

8. Let *loadTimingInfo* be a new document load timing info[p142] with its navigation start time[p142] set to *navigationParams*'s response[p1061]'s timing info's start time.

9. Let *document* be a new Document[p135], with

   type
   : *type*

   content type
   : *contentType*

   origin
   : *navigationParams*'s origin[p1061]

   browsing context[p1046]
   : *browsingContext*

   policy container[p136]
   : *navigationParams*'s policy container[p1061]

permissions policy[p136]
permissionsPolicy
active sandboxing flag set[p959]
navigationParams's final sandboxing flag set[p1061]
opener policy[p136]
navigationParams's cross-origin opener policy[p1061]
load timing info[p141]
loadTimingInfo
was created via cross-origin redirects[p142]
true if *navigationParams*'s response[p1061]'s redirect taint is not "`same-origin`"; otherwise false
during-loading navigation ID for WebDriver BiDi[p136]
navigationParams's id[p1061]
URL
creationURL
current document readiness[p141]
"`loading`"
about base URL[p136]
navigationParams's about base URL[p1062]
allow declarative shadow roots
true
custom element registry
a new `CustomElementRegistry`[p799] object

10. Set *window*'s associated `Document`[p966] to *document*.

11. Set *document*'s internal ancestor origin objects list[p137] to the result of running the internal ancestor origin objects list creation steps[p137] given *document* and *navigationParams*'s `iframe` element referrer policy[p1061].

12. Set *document*'s ancestor origins list[p138] to the result of running the ancestor origins list creation steps[p138] given *document*.

13. Run CSP initialization for a `Document` given *document*. [CSP][p1580]

14. If *navigationParams*'s request[p1061] is non-null:

    1. Set *document*'s referrer[p135] to the empty string.

    2. Let *referrer* be *navigationParams*'s request[p1061]'s referrer.

    3. If *referrer* is a URL record, then set *document*'s referrer[p135] to the serialization of *referrer*.

    Note

    Per Fetch, referrer *will be either a URL record or* "`no-referrer`" *at this point.*

15. If *navigationParams*'s fetch controller[p1061] is not null:

    1. Let *fullTimingInfo* be the result of extracting the full timing info from *navigationParams*'s fetch controller[p1061].

    2. Let *redirectCount* be 0.

    3. If *navigationParams*'s response[p1061]'s redirect taint is "`same-origin`", or all of the following are true:

        - *navigationParams*'s request[p1061]'s client is null, or *navigationParams*'s request[p1061]'s referrer is not "`no-referrer`", and
        - navigation TAO check given *navigationParams*'s response[p1061] and *navigationParams*'s origin[p1061] returns success,

        then set *redirectCount* to *navigationParams*'s request[p1061]'s redirect count.

    4. Create the navigation timing entry for *document*, given *fullTimingInfo*, *redirectCount*, *navigationTimingType*, *navigationParams*'s response[p1061]'s service worker timing info, and *navigationParams*'s response[p1061]'s body info.

16. Create the navigation timing entry for *document*, with *navigationParams*'s response[p1061]'s timing info, *redirectCount*, *navigationParams*'s navigation timing type[p1062], and *navigationParams*'s response[p1061]'s service worker timing info.

17. If *navigationParams*'s response[p1061] has a `` `Refresh`[p1137] `` header:

    1. Let *value* be the isomorphic decoding of the value of the header.

2. Run the shared declarative refresh steps[p204] with *document* and *value*.

We do not currently have a spec for how to handle multiple `Refresh[p1137]` headers. This is tracked as issue #2900.

18. If *navigationParams*'s commit early hints[p1061] is not null, then call *navigationParams*'s commit early hints[p1061] with *document*.
19. Process link headers[p193] given *document*, *navigationParams*'s response[p1061], and "pre-media".
20. If *navigationParams*'s navigable[p1061] is a top-level traversable[p1037], then process the `Speculation-Rules` header[p1133] given *document* and *navigationParams*'s response[p1061].

Note

This is conditional because speculative loads are only considered for top-level traversables[p1128], so it would be wasteful to fetch these rules otherwise.

21. Potentially free deferred fetch quota for *document*.
22. Return *document*.

Example

In this example, the child document is not allowed to use PaymentRequest, despite being same origin-domain[p940] at the time the child document tries to use it. At the time the child document is initialized, only the parent document has set document.domain[p942], and the child document has not.

```
<!-- https://foo.example.com/a.html -->
<!doctype html>
<script>
document.domain = 'example.com';
</script>
<iframe src=b.html></iframe>
```

```
<!-- https://bar.example.com/b.html -->
<!doctype html>
<script>
document.domain = 'example.com'; // This happens after the document is initialized
new PaymentRequest(…); // Not allowed to use
</script>
```

Example

In this example, the child document *is* allowed to use PaymentRequest, despite not being same origin-domain[p940] at the time the child document tries to use it. At the time the child document is initialized, none of the documents have set document.domain[p942] yet so same origin-domain[p940] falls back to a normal same origin[p940] check.

```
<!-- https://example.com/a.html -->
<!doctype html>
<iframe src=b.html></iframe>
<!-- The child document is now initialized, before the script below is run. -->
<script>
document.domain = 'example.com';
</script>
```

```
<!-- https://example.com/b.html -->
<!doctype html>
<script>
new PaymentRequest(…); // Allowed to use
</script>
```

To **populate with html/head/body** given a Document[p135] *document*:

1. Let *html* be the result of creating an element given *document*, "html", and the HTML namespace.

2. Let *head* be the result of creating an element given *document*, "head", and the HTML namespace.
3. Let *body* be the result of creating an element given *document*, "body", and the HTML namespace.
4. Append *html* to *document*.
5. Append *head* to *html*.
6. Append *body* to *html*.

7.5.2 Loading HTML documents §[p11]

To **load an HTML document**, given navigation params[p1061] *navigationParams*:

1. Let *document* be the result of creating and initializing a Document object[p1106] given "html", "text/html", and *navigationParams*.
2. If *document*'s URL is about:blank[p54], then populate with html/head/body[p1109] given *document*.

   Note

   This special case, where even non-initial about:blank[p136] Document[p135]s are synchronously given their element nodes, is necessary for compatibility with deployed content. In other words, it is not compatible to instead go down the "otherwise" branch and feed the empty byte sequence into an HTML parser[p1368] to asynchronously populate document.
3. Otherwise, create an HTML parser[p1368] whose allow declarative shadow roots[p1387] is true and associate it with *document*.

   Each task[p1194] that the networking task source[p1204] places on the task queue[p1193] while fetching runs must then fill the parser's input byte stream[p1374] with the fetched bytes and cause the HTML parser[p1368] to perform the appropriate processing of the input stream.

   The first task[p1194] that the networking task source[p1204] places on the task queue[p1193] while fetching runs must process link headers[p193] given *document*, *navigationParams*'s response[p1061], and "media", after the task has been processed by the HTML parser[p1368].

   Before any script execution occurs, the user agent must wait for scripts may run for the newly-created document[p1101] to be true for *document*.

   Note

   The input byte stream[p1374] converts bytes into characters for use in the tokenizer[p1387]. This process relies, in part, on character encoding information found in the real Content-Type metadata[p102] 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[p1195] on the networking task source[p1204] given *document*'s relevant global object[p1151] to have the parser process the implied EOF character, which eventually causes a load[p1575] event to be fired.
4. Return *document*.

7.5.3 Loading XML documents §[p11]

When faced with displaying an XML file inline, provided navigation params[p1061] *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[p1106] *document*, given "xml", *type*, and *navigationParams*, and return that Document[p135]. They must also create a corresponding XML parser[p1484]. [XML][p1588] [XMLNS][p1588] [RFC7303][p1586] [DOM][p1582]

Note

At the time of writing, the XML specification community had not actually yet specified how XML and the DOM interact.

The first task[p1194] that the networking task source[p1204] places on the task queue[p1193] while fetching runs must process link headers[p193] given *document*, *navigationParams*'s response[p1061], and "media", after the task has been processed by the XML parser[p1484].

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.

Before any script execution occurs, the user agent must wait for scripts may run for the newly-created document[p1101] to be true for the newly-created `Document`[p135].

Once parsing is complete, the user agent must set *document*'s during-loading navigation ID for WebDriver BiDi[p136] to null.

Note

For HTML documents this is reset when parsing is complete, after firing the load event.

Error messages from the parse process (e.g., XML namespace well-formedness errors) may be reported inline by mutating the `Document`[p135].

7.5.4 Loading text documents §[p11]

To **load a text document**, given a navigation params[p1061] *navigationParams* and a string *type*:

1. Let *document* be the result of creating and initializing a `Document` object[p1106] given "`html`", *type*, and *navigationParams*.
2. Set *document*'s parser cannot change the mode flag[p1424] to true.
3. Set *document*'s mode to "`no-quirks`".
4. Create an HTML parser[p1368] 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[p1368]'s tokenizer to the PLAINTEXT state[p1389]. Each task[p1194] that the networking task source[p1204] places on the task queue[p1193] while fetching runs must then fill the parser's input byte stream[p1374] with the fetched bytes and cause the HTML parser[p1368] to perform the appropriate processing of the input stream.

   document's encoding must be set to the character encoding used to decode the document during parsing.

   The first task[p1194] that the networking task source[p1204] places on the task queue[p1193] while fetching runs must process link headers[p193] given *document*, *navigationParams*'s response[p1061], and "`media`", after the task has been processed by the HTML parser[p1368].

   Before any script execution occurs, the user agent must wait for scripts may run for the newly-created document[p1101] to be true for *document*.

   When no more bytes are available, the user agent must queue a global task[p1195] on the networking task source[p1204] given *document*'s relevant global object[p1151] to have the parser process the implied EOF character, which eventually causes a `load`[p1575] event to be fired.
5. User agents may add content to the `head`[p180] element of *document*, e.g., linking to a style sheet, providing script, or giving the document a `title`[p181].

   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.
6. Return *document*.

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*).

7.5.5 Loading `multipart/x-mixed-replace`[p1547] documents §[p11]

To **load a `multipart/x-mixed-replace` document**, given navigation params[p1061] *navigationParams*, source snapshot params[p1060]

sourceSnapshotParams, and origin[p939] *initiatorOrigin*:

1. Parse *navigationParams*'s response[p1061]'s body using the rules for multipart types. [RFC2046][p1585]
2. Let *firstPartNavigationParams* be a copy of *navigationParams*.
3. Set *firstPartNavigationParams*'s response[p1061] to a new response representing the first part of *navigationParams*'s response[p1061]'s body's multipart stream.
4. Let *document* be the result of loading a document[p1088] given *firstPartNavigationParams*, *sourceSnapshotParams*, and *initiatorOrigin*.

   For each additional body part obtained from *navigationParams*'s response[p1061], the user agent must navigate[p1063] *document*'s node navigable[p1036] to *navigationParams*'s request[p1061]'s URL, using *document*, with *response*[p1063] set to *navigationParams*'s response[p1061] and *historyHandling*[p1063] set to "`replace`[p1062]".
5. Return *document*.

For the purposes of algorithms processing these body parts as if they were complete stand-alone resources, the user agent must act as if there were no more bytes for those resources whenever the boundary following the body part is reached.

Note

Thus, `load`[p1575] events (and for that matter `unload`[p1576] events) do fire for each body part loaded.

7.5.6 Loading media documents §[p11 12]

To **load a media document**, given *navigationParams* and a string *type*:

1. Let *document* be the result of creating and initializing a `Document` object[p1106] given "`html`", *type*, and *navigationParams*.
2. Set *document*'s mode to "`no-quirks`".
3. Populate with `html`/`head`/`body`[p1109] given *document*.
4. Append an element *host element* for the media, as described below, to the `body`[p212] element.
5. Set the appropriate attribute of the element *host element*, as described below, to the address of the image, video, or audio resource.
6. User agents may add content to the `head`[p180] element of *document*, or attributes to *host element*, e.g., to link to a style sheet, to provide a script, to give the document a `title`[p181], or to make the media autoplay[p452].
7. Process link headers[p193] given *document*, *navigationParams*'s response[p1061], and "`media`".
8. Act as if the user agent had stopped parsing[p1458] *document*.
9. Return *document*.

The element *host element* to create for the media is the element given in the table below in the second cell of the row whose first cell describes the media. The appropriate attribute to set is the one given by the third cell in that same row.

Type of media	Element for the media	Appropriate attribute
Image	`img`[p359]	`src`[p360]
Video	`video`[p420]	`src`[p433]
Audio	`audio`[p425]	`src`[p433]

Before any script execution occurs, the user agent must wait for scripts may run for the newly-created document[p1101] to be true for the `Document`[p135].

7.5.7 Loading a document for inline content that doesn't have a DOM §[p11 12]

When the user agent is to **create a document to display a user agent page or PDF viewer inline**, provided a navigable[p1035]

navigable, a navigation ID[p1062] *navigationId*, a `NavigationTimingType` *navTimingType*, and a user navigation involvement[p1062] *userInvolvement*, the user agent should:

1. Let *origin* be a new opaque origin[p939].
2. Let *coop* be a new opener policy[p945].
3. Let *coopEnforcementResult* be a new opener policy enforcement result[p948] with

   url[p948]
   response's URL
   origin[p948]
   origin
   opener policy[p948]
   coop
4. Let *navigationParams* be a new navigation params[p1061] with

   id[p1061]
   navigationId
   navigable[p1061]
   navigable
   request[p1061]
   null
   response[p1061]
   a new response
   origin[p1061]
   origin
   fetch controller[p1061]
   null
   commit early hints[p1061]
   null
   COOP enforcement result[p1061]
   coopEnforcementResult
   reserved environment[p1061]
   null
   policy container[p1061]
   a new policy container[p960]
   final sandboxing flag set[p1061]
   an empty set
   `iframe` element referrer policy[p1061]
   the empty string
   opener policy[p1061]
   coop
   navigation timing type[p1062]
   navTimingType
   about base URL[p1062]
   null
   user involvement[p1062]
   userInvolvement
5. Let *document* be the result of creating and initializing a `Document` object[p1106] given "`html`", "`text/html`", and *navigationParams*.
6. Either associate *document* with a custom rendering that is not rendered using the normal `Document`[p135] rendering rules, or mutate *document* until it represents the content the user agent wants to render.
7. Act as if the user agent had stopped parsing[p1458] *document*.

   Note

   This is done to avoid leaking information to the parent page about whether a navigation has been blocked by CSP, or if the response was a network error. In the case where the container is an `iframe`[p404] element, stopping parsing[p1458] causes the iframe load event steps[p408] to run.
8. Return *document*.

Note

Because we ensure the resulting Document *'s origin is opaque, and the resulting* Document *cannot run script with access to the DOM, the existence and properties of this* Document *are not observable to web developer code. This means that most of the above values, e.g., the* text/html *type, do not matter. Similarly, most of the items in* navigationParams *don't have any observable effect, besides preventing the* Document*-creation algorithm from getting confused, and so are set to default values.*

7.5.8 Finishing the loading process

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 node navigable 's container .

   Note

   This will be null in the case where document *is the initial* about:blank Document *in a* frame *or* iframe*, since at the point of browsing context creation which calls this algorithm, the container relationship has not yet been established. (That happens in a subsequent step of create a new child navigable .)*

   The consequence of this is that the following steps do nothing, i.e., we do not fire an asynchronous load *event on the container element for such cases. Instead, a synchronous* load *event is fired in a special initial-insertion case when processing the* iframe *attributes.*
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*.

7.5.9 Unloading documents

A Document has a ***salvageable*** state, which must initially be true, and a **page showing** boolean, which is initially false. The page showing boolean 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).

A Document has a DOMHighResTimeStamp **suspension time**, initially 0.

A Document has a list of **suspended timer handles**, initially empty.

Event loops have a **termination nesting level** counter, which must initially be 0.

Document objects have an **unload counter**, which is used to ignore certain operations while the below algorithms run. Initially, the counter must be set to zero.

To **unload** a Document *oldDocument*, given an optional Document *newDocument*:

1. Assert: this is running as part of a task queued on *oldDocument*'s relevant agent 's event loop .
2. Let *unloadTimingInfo* be a new document unload timing info .
3. If *newDocument* is not given, then set *unloadTimingInfo* to null.

   Note

   In this case there is no new document that needs to know about how long it took oldDocument *to unload.*

4. Otherwise, if *newDocument*'s event loop[p1193] is not *oldDocument*'s event loop[p1193], then the user agent may be unloading[p1114] *oldDocument* in parallel[p44]. In that case, the user agent should set *unloadTimingInfo* to null.

   Note

   In this case newDocument*'s loading is not impacted by how long it takes to unload* oldDocument*, so it would be meaningless to communicate that timing info.*

5. Let *intendToKeepInBfcache* be true if the user agent intends to keep *oldDocument* alive in a session history entry[p1053], such that it can later be used for history traversal[p1054].

   This must be false if *oldDocument* is not *salvageable*[p1114], or if there are any descendants of *oldDocument* which the user agent does not intend to keep alive in the same way (including due to their lack of *salvageability*[p1114]).

6. Let *eventLoop* be *oldDocument*'s relevant agent[p1141]'s event loop[p1193].

7. Increase *eventLoop*'s termination nesting level[p1114] by 1.

8. Increase *oldDocument*'s unload counter[p1114] by 1.

9. If *intendToKeepInBfcache* is false, then set *oldDocument*'s *salvageable*[p1114] state to false.

10. If *oldDocument*'s page showing[p1114] is true:

    1. Set *oldDocument*'s page showing[p1114] to false.

    2. Fire a page transition event[p1029] named `pagehide`[p1576] at *oldDocument*'s relevant global object[p1151] with *oldDocument*'s *salvageable*[p1114] state.

    3. Update the visibility state[p865] of *oldDocument* to "`hidden`".

11. If *unloadTimingInfo* is not null, then set *unloadTimingInfo*'s unload event start time[p142] to the current high resolution time given *newDocument*'s relevant global object[p1151], coarsened given *oldDocument*'s relevant settings object[p1151]'s cross-origin isolated capability[p1144].

12. If *oldDocument*'s *salvageable*[p1114] state is false, then fire an event named `unload`[p1576] at *oldDocument*'s relevant global object[p1151], with *legacy target override flag* set.

13. If *unloadTimingInfo* is not null, then set *unloadTimingInfo*'s unload event end time[p142] to the current high resolution time given *newDocument*'s relevant global object[p1151], coarsened given *oldDocument*'s relevant settings object[p1151]'s cross-origin isolated capability[p1144].

14. Decrease *eventLoop*'s termination nesting level[p1114] by 1.

15. Set *oldDocument*'s suspension time[p1114] to the current high resolution time given *document*'s relevant global object[p1151].

16. Set *oldDocument*'s suspended timer handles[p1114] to the result of getting the keys for the map of active timers[p1256].

17. Set *oldDocument*'s has been scrolled by the user[p1105] to false.

18. Run any unloading document cleanup steps[p1116] for *oldDocument* that are defined by this specification and other applicable specifications[p77].

19. If *oldDocument*'s node navigable[p1036] is a top-level traversable[p1037], build not restored reasons for a top-level traversable and its descendants[p1103] given *oldDocument*'s node navigable[p1036].

20. If *oldDocument*'s *salvageable*[p1114] state is false, then destroy[p1116] *oldDocument*.

21. Decrease *oldDocument*'s unload counter[p1114] by 1.

22. If *newDocument* is given, *newDocument*'s was created via cross-origin redirects[p142] is false, and *newDocument*'s origin is the same[p940] as *oldDocument*'s origin, then set *newDocument*'s previous document unload timing[p141] to *unloadTimingInfo*.

To **unload a document and its descendants**, given a `Document`[p135] *document*, an optional `Document`[p135]-or-null *newDocument* (default null), an optional set of steps *afterAllUnloads*, and an optional set of steps *firePageSwapSteps*:

1. Assert: this is running within *document*'s node navigable[p1036]'s traversable navigable[p1037]'s session history traversal queue[p1037].

2. Let *childNavigables* be *document*'s child navigables[p1039].

3. Let *numberUnloaded* be 0.

4. For each *childNavigable* of *childNavigables* in what order?, queue a global task[p1195] on the navigation and traversal task source[p1204] given *childNavigable*'s active window[p1036] to perform the following steps:

    1. Let *incrementUnloaded* be an algorithm step which increments *numberUnloaded*.

    2. Unload a document and its descendants[p1115] given *childNavigable*'s active document[p1036], null, and *incrementUnloaded*.

5. Wait until *numberUnloaded* equals *childNavigable*'s size.

6. Queue a global task[p1195] on the navigation and traversal task source[p1204] given *document*'s relevant global object[p1151] to perform the following steps:

    1. If *firePageSwapSteps* is given, then run *firePageSwapSteps*.

    2. Unload[p1114] *document*, passing along *newDocument* if it is not null.

    3. If *afterAllUnloads* was given, then run it.

This specification defines the following **unloading document cleanup steps**. Other specifications can define more. Given a `Document`[p135] *document*:

1. Let *window* be *document*'s relevant global object[p1151].

2. For each `WebSocket` object *webSocket* whose relevant global object[p1151] is *window*, make disappear *webSocket*.

    If this affected any `WebSocket` objects, then make document unsalvageable[p1102] given *document* and "`websocket`[p1031]".

3. For each `WebTransport` object *transport* whose relevant global object[p1151] is *window*, run the context cleanup steps given *transport*.

4. If *document*'s *salvageable*[p1114] state is false:

    1. For each `EventSource`[p1285] object *eventSource* whose relevant global object[p1151] is equal to *window*, forcibly close[p1293] *eventSource*.

    2. Clear *window*'s map of active timers[p1256].

It would be better if specification authors sent a pull request to add calls from here into their specifications directly, instead of using the unloading document cleanup steps[p1116] hook, to ensure well-defined cross-specification call order. As of the time of this writing the following specifications are known to have unloading document cleanup steps[p1116], which will be run in an unspecified order: *Fullscreen API*, *Web NFC*, *WebDriver BiDi*, *Compute Pressure*, *File API*, *Media Capture and Streams*, *Picture-in-Picture*, *Screen Orientation*, *Service Workers*, *WebLocks API*, *WebAudio API*, *WebRTC*. [FULLSCREEN][p1582] [WEBNFC][p1588] [WEBDRIVERBIDI][p1588] [COMPUTEPRESSURE][p1579] [FILEAPI][p1582] [MEDIASTREAM][p1584] [PICTUREINPICTURE][p1585] [SCREENORIENTATION][p1586] [SW][p1587] [WEBLOCKS][p1588] [WEBAUDIO][p1587] [WEBRTC][p1588]

Issue #8906 tracks the work to make the order of these steps clear.

7.5.10 Destroying documents §[p1116]

To **destroy** a `Document`[p135] *document*:

1. Assert: this is running as part of a task[p1194] queued on *document*'s relevant agent[p1141]'s event loop[p1193].

2. Abort[p1117] *document*.

3. Set *document*'s *salvageable*[p1114] state to false.

4. Run any unloading document cleanup steps[p1116] for *document* that are defined by this specification and other applicable specifications[p77].

5. Remove any tasks[p1194] whose document[p1194] is *document* from any task queue[p1193] (without running those tasks).

6. Set *document*'s browsing context to null.
7. Set *document*'s node navigable's active session history entry's document state's document 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*.

Note

Even after destruction, the `Document` object itself might still be accessible to script, in the case where we are destroying a child navigable.

To **destroy a document and its descendants** given a `Document` *document* and an optional set of steps *afterAllDestruction*, perform the following steps in parallel:

1. If *document* is not fully active:
    1. Let *reason* be a string from user-agent specific blocking reasons. If none apply, then let *reason* be "`masked`".
    2. Make document unsalvageable given *document* and *reason*.
    3. If *document*'s node navigable is a top-level traversable, build not restored reasons for a top-level traversable and its descendants given *document*'s node navigable.
2. Let *childNavigables* be *document*'s child navigables.
3. Let *numberDestroyed* be 0.
4. For each *childNavigable* of *childNavigables* in what order?, queue a global task on the navigation and traversal task source given *childNavigable*'s active window to perform the following steps:
    1. Let *incrementDestroyed* be an algorithm step which increments *numberDestroyed*.
    2. Destroy a document and its descendants given *childNavigable*'s active document and *incrementDestroyed*.
5. Wait until *numberDestroyed* equals *childNavigable*'s size.
6. Queue a global task on the navigation and traversal task source given *document*'s relevant global object to perform the following steps:
    1. Destroy *document*.
    2. If *afterAllDestruction* was given, then run it.

7.5.11 Aborting a document load

To **abort** a `Document` *document*:

1. Assert: this is running as part of a task queued on *document*'s relevant agent's event loop.
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 make document unsalvageable given *document* and "`fetch`".
3. If *document*'s during-loading navigation ID for WebDriver BiDi is non-null:
    1. Invoke WebDriver BiDi navigation aborted with *document*'s node navigable and a new WebDriver BiDi navigation status whose id is *document*'s during-loading navigation ID for WebDriver BiDi, status is "`canceled`", and url is *document*'s URL.
    2. Set *document*'s during-loading navigation ID for WebDriver BiDi to null.
4. If *document* has an active parser:

1. Set *document*'s active parser was aborted to true.
2. Abort that parser.
3. Make document unsalvageable given *document* and "parser-aborted".

To **abort a document and its descendants** given a Document *document*:

1. Assert: this is running as part of a task queued on *document*'s relevant agent's event loop.
2. Let *descendantNavigables* be *document*'s descendant navigables.
3. For each *descendantNavigable* of *descendantNavigables* in what order?, queue a global task on the navigation and traversal task source given *descendantNavigable*'s active window to perform the following steps:
    1. Abort *descendantNavigable*'s active document.
    2. If *descendantNavigable*'s active document's *salvageable* is false, then set *document*'s *salvageable* to false.
4. Abort *document*.

To **stop loading** a navigable *navigable*:

1. Let *document* be *navigable*'s active document.
2. If *document*'s unload counter is 0, and *navigable*'s ongoing navigation is a navigation ID, then set the ongoing navigation for *navigable* to null.

    Note

    This will have the effect of aborting any ongoing navigations of navigable*, since at certain points during navigation, changes to the ongoing navigation will cause further work to be abandoned.*

3. Abort a document and its descendants given *document*.

Through their user interface, user agents also allow stopping traversals, i.e. cases where the ongoing navigation is "traversal". The above algorithm does not account for this. (On the other hand, user agents do not allow window.stop() to stop traversals, so the above algorithm is correct for that caller.) See issue #6905.

7.6 Speculative loading §

Speculative loading is the practice of performing navigation actions, such as prefetching, ahead of navigation starting. This makes subsequent navigations faster.

Developers can initiate speculative loads by using speculation rules. User agents might also perform speculative loads in certain implementation-defined scenarios, such as typing into the address bar.

7.6.1 Speculation rules §

Speculation rules are how developers instruct the browser about speculative loading operations that the developer believes will be beneficial. They are delivered as JSON documents, via either:

- inline script elements with their type attribute set to "speculationrules"; or
- resources fetched from a URL specified in the `Speculation-Rules` HTTP response header.

¶ Example

The following JSON document is parsed into a speculation rule set specifying a number of desired conditions for the user agent

to start a referrer-initiated navigational prefetch:

```
{
  "prefetch": [
    {
      "urls": ["/chapters/5"]
    },
    {
      "eagerness": "moderate",
      "where": {
        "and": [
          { "href_matches": "/*" },
          { "not": { "selector_matches": ".no-prefetch" } }
        ]
      }
    }
  ]
}
```

A JSON document representing a speculation rule set[p1120] must meet the following **speculation rule set authoring requirements**:

- It must be valid JSON. [JSON][p1583]
- The JSON must represent a JSON object, with at most three keys "`tag`", "`prefetch`" and "`prerender`".

  Note

  In this standard, "`prerender`" *is optionally converted to* "`prefetch`" *at parse time*[p1122]*. Some implementations might implement different behavior for prerender, as specified in* Prerendering Revamped. *[PRERENDERING-REVAMPED]*[p1585]
- The value corresponding to the "`tag`" key, if present, must be a speculation rule tag[p1121].
- The values corresponding to the "`prefetch`" and "`prerender`" keys, if present, must be arrays of valid speculation rules[p1119].

A **valid speculation rule** is a JSON object that meets the following requirements:

- It must have at most the following keys: "`source`", "`urls`", "`where`", "`relative_to`", "`eagerness`", "`referrer_policy`", "`tag`", "`requires`", "`expects_no_vary_search`", or "`target_hint`".

  Note

  In this standard, "`target_hint`" *is ignored*[p1123].
- The value corresponding to the "`source`" key, if present, must be either "`list`" or "`document`".
- If the value corresponding to the "`source`" key is "`list`", then the "`urls`" key must be present, and the "`where`" key must be absent.
- If the value corresponding to the "`source`" key is "`document`", then the "`urls`" key must be absent.
- The "`urls`" and "`where`" keys must not both be present.
- If the value corresponding to the "`source`" key is "`document`" or the "`where`" key is present, then the "`relative_to`" key must be absent.
- The value corresponding to the "`urls`" key, if present, must be an array of valid URL strings.
- The value corresponding to the "`where`" key, if present, must be a valid document rule predicate[p1120].
- The value corresponding to the "`relative_to`" key, if present, must be either "`ruleset`" or "`document`".
- The value corresponding to the "`eagerness`" key, if present, must be a speculation rule eagerness[p1121].
- The value corresponding to the "`referrer_policy`" key, if present, must be a referrer policy.
- The value corresponding to the "`tag`" key, if present, must be a speculation rule tag[p1121].

- The value corresponding to the "`requires`" key, if present, must be an array of speculation rule requirements[p1121].
- The value corresponding to the "`expects_no_vary_search`" key, if present, must be a string that is parseable as a `No-Vary-Search` header value.

A **valid document rule predicate** is a JSON object that meets the following requirements:

- It must contain exactly one of the keys "`and`", "`or`", "`not`", "`href_matches`", or "`selector_matches`".
- It must not contain any keys apart from the above or "`relative_to`".
- If it contains the key "`relative_to`", then it must also contain the key "`href_matches`".
- The value corresponding to the "`relative_to`" key, if present, must be either "`ruleset`" or "`document`".
- The value corresponding to the "`and`" or "`or`" keys, if present, must be arrays of valid document rule predicates[p1120].
- The value corresponding to the "`not`" key, if present, must be a valid document rule predicate[p1120].
- The value corresponding to the "`href_matches`" key, if present, must be either a valid URL pattern input[p1120] or an array of valid URL pattern inputs[p1120].
- The value corresponding to the "`selector_matches`" key, if present, must be either a string matching `<selector-list>` or an array of strings that match `<selector-list>`.

A **valid URL pattern input** is either:

- a scalar value string that can be successfully parsed as a URL pattern constructor string, or;
- a JSON object whose keys are drawn from the members of the `URLPatternInit` dictionary and whose values are scalar value strings.

7.6.1.1 Data model §[p1120]

A **speculation rule set** is a struct with the following items:

- **prefetch rules**, a list of speculation rules[p1120], initially empty

Note

In the future, other rules will be possible, e.g., prerender rules. See Prerendering Revamped *for such not-yet-accepted extensions. [PRERENDERING-REVAMPED][p1585]*

A **speculation rule** is a struct with the following items:

- **URLs**, an ordered set of URLs
- **predicate**, a document rule predicate[p1120] or null
- **eagerness**, a speculation rule eagerness[p1121]
- **referrer policy**, a referrer policy
- **tags**, an ordered set of speculation rule tags[p1121]
- **requirements**, an ordered set of speculation rule requirements[p1121]
- **No-Vary-Search hint**, a URL search variance

A **document rule predicate** is one of the following:

- a document rule conjunction[p1121];
- a document rule disjunction[p1121];
- a document rule negation[p1121];

- a document rule URL pattern predicate[p1121]; or
- a document rule selector predicate[p1121].

A **document rule conjunction** is a struct with the following items:

- **clauses**, a list of document rule predicates[p1120]

A **document rule disjunction** is a struct with the following items:

- **clauses**, a list of document rule predicates[p1120]

A **document rule negation** is a struct with the following items:

- **clause**, a document rule predicate[p1120]

A **document rule URL pattern predicate** is a struct with the following items:

- **patterns**, a list of URL patterns

A **document rule selector predicate** is a struct with the following items:

- **selectors**, a list of selectors

A **speculation rule eagerness** is one of the following strings:

"`immediate`"
: The developer believes that performing the associated speculative loads is very likely to be worthwhile, and they might also expect that load to require significant lead time to complete. User agents should usually enact the speculative load candidate as soon as practical, subject only to considerations such as user preferences, device conditions, and resource limits.

"`eager`"
: User agents should enact the speculative load candidate on even a slight suggestion that the user may navigate to this URL in the future. For instance, the user might have moved the cursor toward a link or hovered it, even momentarily, or paused scrolling when the link is one of the more prominent ones in the viewport. The author is seeking to capture as many navigations as possible, as early as possible.

"`moderate`"
: User agents should enact the candidate if user behavior suggests the user may navigate to this URL in the near future. For instance, the user might have scrolled a link into the viewport and shown signs of being likely to click it, e.g., by moving the cursor over it for some time. The developer is seeking a balance between "`eager`[p1121]" and "`conservative`[p1121]".

"`conservative`"
: User agents should enact the candidate only when the user is very likely to navigate to this URL at any moment. For instance, the user might have begun to interact with a link. The developer is seeking to capture some of the benefits of speculative loading with a fairly small tradeoff of resources.

A speculation rule eagerness[p1121] *A* is **less eager** than another speculation rule eagerness[p1121] *B* if *A* follows *B* in the above list.

A speculation rule eagerness[p1121] *A* is **at least as eager** as another speculation rule eagerness[p1121] *B* if *A* is not less eager[p1121] than *B*.

A **speculation rule tag** is either an ASCII string whose code points are all in the range U+0020 to U+007E inclusive, or null.

Note

This code point range restriction ensures the value can be sent in an HTTP header with no escaping or modification.

A **speculation rule requirement** is the string "`anonymous-client-ip-when-cross-origin`".

Note

In the future, more possible requirements might be defined.

7.6.1.2 Parsing §p1122

Note

Since speculative loading is a progressive enhancement, this standard is fairly conservative in its parsing behavior. In particular, unknown keys or invalid values usually cause parsing failure, since it is safer to do nothing than to possibly misinterpret a speculation rule.

That said, parsing failure for a single speculation rule still allows other speculation rules to be processed. It is only in the case of top-level misconfiguration that the entire speculation rule set is discarded.

To **parse a speculation rule set string** given a string *input*, a `Document`p135 *document*, and a URL *baseURL*:

1. Let *parsed* be the result of parsing a JSON string to an Infra value given *input*.
2. If *parsed* is not a map, then throw a `TypeError` indicating that the top-level value needs to be a JSON object.
3. Let *result* be a new speculation rule setp1120.
4. Let *tag* be null.
5. If *parsed*["`tag`"] exists:
    1. If *parsed*["`tag`"] is not a speculation rule tagp1121, then throw a `TypeError` indicating that the speculation rule tag is invalid.
    2. Set *tag* to *parsed*["`tag`"].
6. Let *typesToTreatAsPrefetch* be « "`prefetch`" ».
7. The user agent may append "`prerender`" to *typesToTreatAsPrefetch*.

¶p1122 Note

Since this specification only includes prefetching, this allows user agents to treat requests for prerendering as requests for prefetching. User agents which implement prerendering, per the Prerendering Revamped *specification, will instead interpret these as prerender requests. [PRERENDERING-REVAMPED]p1585*

8. For each *type* of *typesToTreatAsPrefetch*:
    1. If *parsed*[*type*] exists:
        1. If *parsed*[*type*] is a list, then for each *rule* of *parsed*[*type*]:
            1. Let *rule* be the result of parsing a speculation rulep1122 given *rule*, *tag*, *document*, and *baseURL*.
            2. If *rule* is null, then continue.
            3. Append *rule* to *result*'s prefetch rulesp1120.
        2. Otherwise, the user agent may report a warning to the console indicating that the rules list for *type* needs to be a JSON array.
9. Return *result*.

To **parse a speculation rule** given a map *input*, a speculation rule tagp1121 *rulesetLevelTag*, a `Document`p135 *document*, and a URL *baseURL*:

1. If *input* is not a map:
    1. The user agent may report a warning to the console indicating that the rule needs to be a JSON object.
    2. Return null.
2. If *input* has any key other than "`source`", "`urls`", "`where`", "`relative_to`", "`eagerness`", "`referrer_policy`", "`tag`", "`requires`", "`expects_no_vary_search`", or "`target_hint`":
    1. The user agent may report a warning to the console indicating that the rule has unrecognized keys.
    2. Return null.

¶p11 23 **Note**

"target_hint" *has no impact on the processing model in this standard. However, implementations of* Prerendering Revamped *can use it for prerendering rules, and so requiring user agents to fail parsing such rules would be counterproductive. [PRERENDERING-REVAMPED]*p1585*.*

3. Let *source* be null.
4. If *input*["source"] exists, then set *source* to *input*["source"].
5. Otherwise, if *input*["urls"] exists and *input*["where"] does not exist, then set *source* to "list".
6. Otherwise, if *input*["where"] exists and *input*["urls"] does not exist, then set *source* to "document".
7. If *source* is neither "list" nor "document":
    1. The user agent may report a warning to the console indicating that a source could not be inferred or an invalid source was specified.
    2. Return null.
8. Let *urls* be an empty list.
9. Let *predicate* be null.
10. If *source* is "list":
    1. If *input*["where"] exists:
        1. The user agent may report a warning to the console indicating that there were conflicting sources for this rule.
        2. Return null.
    2. If *input*["relative_to"] exists:
        1. If *input*["relative_to"] is neither "ruleset" nor "document":
            1. The user agent may report a warning to the console indicating that the supplied relative-to value was invalid.
            2. Return null.
        2. If *input*["relative_to"] is "document", then set *baseURL* to *document*'s document base URL*p101*.
    3. If *input*["urls"] does not exist or is not a list:
        1. The user agent may report a warning to the console indicating that the supplied URL list was invalid.
        2. Return null.
    4. For each *urlString* of *input*["urls"]:
        1. If *urlString* is not a string:
            1. The user agent may report a warning to the console indicating that the supplied URL must be a string.
            2. Return null.
        2. Let *parsedURL* be the result of URL parsing *urlString* with *baseURL*.
        3. If *parsedURL* is failure, or *parsedURL*'s scheme is not an HTTP(S) scheme:
            1. The user agent may report a warning to the console indicating that the supplied URL string was unparseable.
            2. Continue.
        4. Append *parsedURL* to *urls*.
11. If *source* is "document":

1. If *input*["urls"] or *input*["relative_to"] exists:

    1. The user agent may report a warning to the console indicating that there were conflicting sources for this rule.

    2. Return null.

2. If *input*["where"] does not exist, then set *predicate* to a document rule conjunction[p1121] whose clauses[p1121] is an empty list.

    Note

    Such a predicate will match all links.

3. Otherwise, set *predicate* to the result of parsing a document rule predicate[p1125] given *input*["where"], *document*, and *baseURL*.

4. If *predicate* is null, then return null.

12. Let *eagerness* be "immediate[p1121]" if *source* is "list"; otherwise, "conservative[p1121]".

13. If *input*["eagerness"] exists:

    1. If *input*["eagerness"] is not a speculation rule eagerness[p1121]:

        1. The user agent may report a warning to the console indicating that the eagerness was invalid.

        2. Return null.

    2. Set *eagerness* to *input*["eagerness"].

14. Let *referrerPolicy* be the empty string.

15. If *input*["referrer_policy"] exists:

    1. If *input*["referrer_policy"] is not a referrer policy:

        1. The user agent may report a warning to the console indicating that the referrer policy was invalid.

        2. Return null.

    2. Set *referrerPolicy* to *input*["referrer_policy"].

16. Let *tags* be an empty ordered set.

17. If *rulesetLevelTag* is not null, then append *rulesetLevelTag* to *tags*.

18. If *input*["tag"] exists:

    1. If *input*["tag"] is not a speculation rule tag[p1121]:

        1. The user agent may report a warning to the console indicating that the tag was invalid.

        2. Return null.

    2. Append *input*["tag"] to *tags*.

19. If *tags* is empty, then append null to *tags*.

20. Assert: *tags*'s size is either 1 or 2.

21. Let *requirements* be an empty ordered set.

22. If *input*["requires"] exists:

    1. If *input*["requires"] is not a list:

        1. The user agent may report a warning to the console indicating that the requirements were not understood.

        2. Return null.

    2. For each *requirement* of *input*["requires"]:

1. If *requirement* is not a speculation rule requirement[p1121]:

    1. The user agent may report a warning to the console indicating that the requirement was not understood.

    2. Return null.

2. Append *requirement* to *requirements*.

23. Let *noVarySearchHint* be the default URL search variance.

24. If *input*["`expects_no_vary_search`"] exists:

    1. If *input*["`expects_no_vary_search`"] is not a string:

        1. The user agent may report a warning to the console indicating that the `` `No-Vary-Search` `` hint was invalid.

        2. Return null.

    2. Set *noVarySearchHint* to the result of parsing a URL search variance given *input*["`expects_no_vary_search`"].

25. Return a speculation rule[p1120] with:

    URLs[p1120]
    urls
    predicate[p1120]
    predicate
    eagerness[p1120]
    eagerness
    referrer policy[p1120]
    referrerPolicy
    tags[p1120]
    tags
    requirements[p1120]
    requirements
    No-Vary-Search hint[p1120]
    noVarySearchHint

To **parse a document rule predicate** given a value *input*, a `Document`[p135] *document*, and a URL *baseURL*:

1. If *input* is not a map:

    1. The user agent may report a warning to the console indicating that the document rule predicate was invalid.

    2. Return null.

2. If *input* does not contain exactly one of "`and`", "`or`", "`not`", "`href_matches`", or "`selector_matches`":

    1. The user agent may report a warning to the console indicating that the document rule predicate was empty or ambiguous.

    2. Return null.

3. Let *predicateType* be the single key found in the previous step.

4. If *predicateType* is "`and`" or "`or`":

    1. If *input* has any key other than *predicateType*:

        1. The user agent may report a warning to the console indicating that the document rule predicate had unexpected extra options.

        2. Return null.

    2. If *input*[*predicateType*] is not a list:

        1. The user agent may report a warning to the console indicating that the document rule predicate had an invalid clause list.

        2. Return null.

3. Let *clauses* be an empty list.
4. For each *rawClause* of *input*[*predicateType*]:
    1. Let *clause* be the result of parsing a document rule predicate[p1125] given *rawClause*, *document*, and *baseURL*.
    2. If *clause* is null, then return null.
    3. Append *clause* to *clauses*.
5. If *predicateType* is "`and`", then return a document rule conjunction[p1121] whose clauses[p1121] is *clauses*.
6. Return a document rule disjunction[p1121] whose clauses[p1121] is *clauses*.

5. If *predicateType* is "`not`":
    1. If *input* has any key other than "`not`":
        1. The user agent may report a warning to the console indicating that the document rule predicate had unexpected extra options.
        2. Return null.
    2. Let *clause* be the result of parsing a document rule predicate[p1125] given *input*[*predicateType*], *document*, and *baseURL*.
    3. If *clause* is null, then return null.
    4. Return a document rule negation[p1121] whose clause[p1121] is *clause*.
6. If *predicateType* is "`href_matches`":
    1. If *input* has any key other than "`href_matches`" or "`relative_to`":
        1. The user agent may report a warning to the console indicating that the document rule predicate had unexpected extra options.
        2. Return null.
    2. If *input*["`relative_to`"] exists:
        1. If *input*["`relative_to`"] is neither "`ruleset`" nor "`document`":
            1. The user agent may report a warning to the console indicating that the supplied relative-to value was invalid.
            2. Return null.
        2. If *input*["`relative_to`"] is "`document`", then set *baseURL* to *document*'s document base URL[p101].
    3. Let *rawPatterns* be *input*["`href_matches`"].
    4. If *rawPatterns* is not a list, then set *rawPatterns* to « *rawPatterns* ».
    5. Let *patterns* be an empty list.
    6. For each *rawPattern* of *rawPatterns*:
        1. Let *pattern* be the result of building a URL pattern from an Infra value given *rawPattern* and *baseURL*. If this step throws an exception, catch the exception and set *pattern* to null.
        2. If *pattern* is null:
            1. The user agent may report a warning to the console indicating that the supplied URL pattern was invalid.
            2. Return null.
        3. Append *pattern* to *patterns*.
    7. Return a document rule URL pattern predicate[p1121] whose patterns[p1121] is *patterns*.

7. If *predicateType* is "`selector_matches`":
    1. If *input* has any key other than "`selector_matches`":
        1. The user agent may report a warning to the console indicating that the document rule predicate had unexpected extra options.
        2. Return null.
    2. Let *rawSelectors* be *input*["`selector_matches`"].
    3. If *rawSelectors* is not a list, then set *rawSelectors* to « *rawSelectors* ».
    4. Let *selectors* be an empty list.
    5. For each *rawSelector* of *rawSelectors*:
        1. Let *parsedSelectorList* be failure.
        2. If *rawSelector* is a string, then set *parsedSelectorList* to the result of parsing a selector given *rawSelector*.
        3. If *parsedSelectorList* is failure:
            1. The user agent may report a warning to the console indicating that the supplied selector list was invalid.
            2. Return null.
        4. For each *selector* of *parsedSelectorList*, append *selector* to *selectors*.
    6. Return a document rule selector predicate[p1121] whose selectors[p1121] is *selectors*.
8. Assert: this step is never reached, as one of the previous branches was taken.

7.6.1.3 Processing model §[p1127]

A **speculative load candidate** is a struct with the following items:

- **URL**, a URL
- **No-Vary-Search hint**, a URL search variance
- **eagerness**, a speculation rule eagerness[p1121]
- **referrer policy**, a referrer policy
- **tags**, an ordered set of speculation rule tags[p1121]

A **prefetch candidate** is a speculative load candidate[p1127] with the following additional item:

- **anonymization policy**, a prefetch IP anonymization policy[p1127]

A **prefetch IP anonymization policy** is either null or a cross-origin prefetch IP anonymization policy[p1127].

A **cross-origin prefetch IP anonymization policy** is a struct whose single item is its **origin**, an origin[p939].

A speculative load candidate[p1127] *candidateA* is **redundant with** another speculative load candidate[p1127] *candidateB* if the following steps return true:

1. If *candidateA*'s No-Vary-Search hint[p1127] is not equal to *candidateB*'s No-Vary-Search hint[p1127], then return false.
2. If *candidateA*'s URL[p1127] is not equivalent modulo search variance to *candidateB*'s URL[p1127] given *candidateA*'s No-Vary-Search hint[p1127], then return false.
3. Return true.

¶ p11 28

Note

The requirement that the No-Vary-Search hints p1127 be equivalent is somewhat strict. It means that some cases which could theoretically be treated as matching, are not treated as such. Thus, redundant speculative loads could happen.

However, allowing more lenient matching makes the check no longer an equivalence relation, and producing such matches would require an implementation strategy that does a full comparison, instead of a simpler one using normalized URL keys. This is in line with the best practices for server operators, and attendant HTTP cache implementation notes, in No Vary Search § 6 Comparing.

In practice, we do not expect this to cause redundant speculative loads, since server operators and the corresponding speculation rules-writing web developers will follow best practices and use static `No-Vary-Search` header values/speculation rule hints.

¶ p11 28

Example

Consider three speculative load candidates p1127:

1. *A* has a URL p1127 of `https://example.com?a=1&b=1` and a No-Vary-Search hint p1127 parsed from `params=("a")`.
2. *B* has a URL p1127 of `https://example.com?a=2&b=1` and a No-Vary-Search hint p1127 parsed from `params=("b")`.
3. *C* has a URL p1127 of `https://example.com?a=2&b=2` and a No-Vary-Search hint p1127 parsed from `params=("a")`.

With the current definition of redundant with p1127, none of these candidates are redundant with each other. A speculation rule set p1120 which contained all three could cause three separate speculative loads.

A definition which did not require equivalent No-Vary-Search hints p1127 could consider *A* and *B* to match (using *A*'s No-Vary-Search hint p1127), and *B* and *C* to match (using *B*'s No-Vary-Search hint p1127). But it could not consider *A* and *C* to match, so it would not be transitive, and thus not an equivalence relation.

Every `Document` p135 has **speculation rule sets**, a list of speculation rule sets p1120, initially empty.

Every `Document` p135 has a **consider speculative loads microtask queued**, a boolean, initially false.

To **consider speculative loads** for a `Document` p135 *document*:

1. If *document*'s node navigable p1036 is not a top-level traversable p1037, then return.

   Note

   Supporting speculative loads into child navigables p1039 has some complexities and is not currently defined. It might be possible to define it in the future.

2. If *document*'s consider speculative loads microtask queued p1128 is true, then return.
3. Set *document*'s consider speculative loads microtask queued p1128 to true.
4. Queue a microtask p1195 given *document* to run the following steps:
   1. Set *document*'s consider speculative loads microtask queued p1128 to false.
   2. Run the inner consider speculative loads steps p1129 for *document*.

In addition to the call sites explicitly given in this standard:

- When style recalculation would cause selector matching results to change, the user agent must consider speculative loads p1128 for the relevant `Document` p135.
- When the user indicates interest in hyperlinks p313, in one of the implementation-defined ways that the user agent uses to implement the speculation rule eagerness p1121 heuristics, the user agent may consider speculative loads p1128 for the hyperlink's node document.

  Example

  For example, a user agent which implements "`conservative` p1121" eagerness by watching for `pointerdown` events would want to consider speculative loads p1128 as part of reacting to such events.

¶ p1129

Note

In this standard, every call to consider speculative loads p1128 is given just a Document p135, and the algorithm re-computes all possible candidates in a stateless way. A real implementation would likely cache previous computations, and pass along information from the call site to make updates more efficient. For example, if an a p267 element's href p314 attribute is changed, that specific element could be passed along in order to update only the related speculative load candidate p1127.

Note that because of how consider speculative loads p1128 queues a microtask, by the time the inner consider speculative loads steps p1129 are run, multiple updates (or cancelations p1130) might be processed together.

The **inner consider speculative loads steps** for a Document p135 *document* are:

1. If *document* is not fully active p1051, then return.
2. Let *prefetchCandidates* be an empty list.
3. For each *ruleSet* of *document*'s speculation rule sets p1128:
    1. For each *rule* of *ruleSet*'s prefetch rules p1120:
        1. Let *anonymizationPolicy* be null.
        2. If *rule*'s requirements p1120 contains "`anonymous-client-ip-when-cross-origin`", then set *anonymizationPolicy* to a cross-origin prefetch IP anonymization policy p1127 whose origin p1127 is *document*'s origin.
        3. For each *url* of *rule*'s URLs p1120:
            1. Let *referrerPolicy* be the result of computing a speculative load referrer policy p1131 given *rule* and null.
            2. Append a new prefetch candidate p1127 with

               URL p1127
               : *url*

               No-Vary-Search hint p1127
               : *rule*'s No-Vary-Search hint p1120

               eagerness p1127
               : *rule*'s eagerness p1120

               referrer policy p1127
               : *referrerPolicy*

               tags p1127
               : *rule*'s tags p1120

               anonymization policy p1127
               : *anonymizationPolicy*

               to *prefetchCandidates*.
        4. If *rule*'s predicate p1120 is not null:
            1. Let *links* be the result of finding matching links p1131 given *document* and *rule*'s predicate p1120.
            2. For each *link* of *links*:
                1. Let *referrerPolicy* be the result of computing a speculative load referrer policy p1131 given *rule* and *link*.
                2. Append a new prefetch candidate p1127 with

                   URL p1127
                   : *link*'s url p316

                   No-Vary-Search hint p1127
                   : *rule*'s No-Vary-Search hint p1120

                   eagerness p1127
                   : *rule*'s eagerness p1120

                   referrer policy p1127
                   : *referrerPolicy*

                   tags p1127
                   : *rule*'s tags p1120

anonymization policy[p1127]
anonymizationPolicy

to *prefetchCandidates*.

4. For each *prefetchRecord* of *document*'s prefetch records:

    1. If *prefetchRecord*'s source is not "`speculation rules`", then continue.

    2. Assert: *prefetchRecord*'s state is not "`canceled`".

    3. If *prefetchRecord* is not still being speculated given *prefetchCandidates*, then cancel and discard *prefetchRecord* given *document*.

5. Let *prefetchCandidateGroups* be an empty list.

6. For each *candidate* of *prefetchCandidates*:

    1. Let *group* be « *candidate* ».

    2. Extend *group* with all items in *prefetchCandidates*, apart from *candidate* itself, which are redundant with[p1127] *candidate* and whose eagerness[p1127] is at least as eager[p1121] as *candidate*'s eagerness[p1127].

    3. If *prefetchCandidateGroups* contains another group whose items are the same as *group*, ignoring order, then continue.

    4. Append *group* to *prefetchCandidateGroups*.

¶[p1130] **Example**

The following speculation rules generate two redundant[p1127] prefetch candidates[p1127]:

```
{
  "prefetch": [
    {
      "tag": "a",
      "urls": ["next.html"]
    },
    {
      "tag": "b",
      "urls": ["next.html"],
      "referrer_policy": "no-referrer"
    }
  ]
}
```

This step will create a single group containing them both, in the given order. (The second pass through will not create a group, since its contents would be the same as the first group, just in a different order.) This means that if the user agent chooses to execute the "may" step below to enact the group, it will enact the first candidate, and ignore the second. Thus, the request will be made with the default referrer policy, instead of using "`no-referrer`".

However, the collect tags from speculative load candidates[p1131] algorithm will collect tags from both candidates in the group, so the `Sec-Speculation-Tags`[p1133] header value will be `"a", "b"`. This indicates to server operators that either rule could have caused the speculative load.

7. For each *group* of *prefetchCandidateGroups*:

    1. The user agent may run the following steps:

        1. Let *prefetchCandidate* be *group*[0].

        2. Let *tagsToSend* be the result of collecting tags from speculative load candidates[p1131] given *group*.

        3. Let *prefetchRecord* be a new prefetch record with

            source
            "`speculation rules`"
            URL
            prefetchCandidate's URL[p1127]

No-Vary-Search hint
prefetchCandidate's No-Vary-Search hint[p1127]
referrer policy
prefetchCandidate's referrer policy[p1127]
anonymization policy
prefetchCandidate's anonymization policy[p1127]
tags
tagsToSend

4. Start a referrer-initiated navigational prefetch given *document* and *prefetchRecord*.

When deciding whether to execute this "may" step, user agents should consider *prefetchCandidate*'s eagerness[p1127], in accordance to the current behavior of the user and the definitions of speculation rule eagerness[p1121].

prefetchCandidate's No-Vary-Search hint[p1127] can also be useful in implementing the heuristics defined for the speculation rule eagerness[p1121] values. For example, a user hovering of a link whose url[p316] is equivalent modulo search variance to *prefetchCandidate*'s URL[p1127] given *prefetchCandidate*'s No-Vary-Search hint[p1127] could indicate to the user agent that performing this step would be useful.

When deciding whether to execute this "may" step, user agents should prioritize user preferences (express or implied, such as data-saver or battery-saver modes) over the eagerness supplied by the web developer.

To **compute a speculative load referrer policy** given a speculation rule[p1120] *rule* and an a[p267] element, area[p489] element, or null *link*:

1. If *rule*'s referrer policy[p1120] is not the empty string, then return *rule*'s referrer policy[p1120].
2. If *link* is null, then return the empty string.
3. Return *link*'s hyperlink referrer policy[p322].

To **collect tags from speculative load candidates** given a list of speculative load candidates[p1127] *candidates*:

1. Let *tags* be an empty ordered set.
2. For each *candidate* of *candidates*:
    1. For each *tag* of *candidate*'s tags[p1127]: append *tag* to *tags*.
3. Sort in ascending order *tags*, with *tagA* being less than *tagB* if *tagA* is null, or if *tagA* is code unit less than *tagB*.
4. Return *tags*.

To **find matching links** given a Document[p135] *document* and a document rule predicate[p1120] *predicate*:

1. Let *links* be an empty list.
2. For each shadow-including descendant *descendant* of *document*, in shadow-including tree order:
    1. If *descendant* is not an a[p267] or area[p489] element with an href[p314] attribute, then continue.
    2. If *descendant* is not being rendered[p1488] or is part of skipped contents, then continue.

    Note

    Such links, though present in document*, aren't available for the user to interact with, and thus are unlikely to be good candidates. In addition, they might not have their style or layout computed, which might make selector matching less efficient in user agents which skip some or all of that work for these elements.*

    3. If *descendant*'s url[p316] is null, or its scheme is not an HTTP(S) scheme, then continue.
    4. If *predicate* matches[p1131] *descendant*, then append *descendant* to *links*.
3. Return *links*.

A document rule predicate[p1120] *predicate* **matches** an a[p267] or area[p489] element *el* if the following steps return true, switching on *predicate*'s type:

↪ **document rule conjunction**[p1121]

1. For each *clause* of *predicate*'s clauses[p1121]:
    1. If *clause* does not match[p1131] *el*, then return false.
2. Return true.

↪ **document rule disjunction**[p1121]

1. For each *clause* of *predicate*'s clauses[p1121]:
    1. If *clause* matches[p1131] *el*, then return true.
2. Return false.

↪ **document rule negation**[p1121]

1. If *predicate*'s clause[p1121] matches[p1131] *el*, then return false.
2. Return true.

↪ **document rule URL pattern predicate**[p1121]

1. For each *pattern* of *predicate*'s patterns[p1121]:
    1. If performing a match given *pattern* and *el*'s url[p316] gives a non-null value, then return true.
2. Return false.

↪ **document rule selector predicate**[p1121]

1. For each *selector* of *predicate*'s selectors[p1121]:
    1. If performing a match given *selector* and *el* with the scoping root set to *el*'s root returns success, then return true.
2. Return false.

Speculation rules features use the **speculation rules task source**, which is a task source[p1194].

Note

Because speculative loading is generally less important than processing tasks for the purpose of the current document, implementations might give tasks[p1194] enqueued here an especially low priority.

7.6.2 Navigational prefetching §[p11 32]

For now, the navigational prefetching process is defined in the *Prefetch* specification. Moving it into this standard is tracked in issue #11123. [PREFETCH][p1585]

This standard refers to the following concepts defined there:

- **prefetch record**, and its items **source**, **URL**, **No-Vary-Search hint**, **referrer policy**, **anonymization policy**, **tags**, and **state**
- **cancel and discard**
- **still being speculated**
- **prefetch records**
- **start a referrer-initiated navigational prefetch**

7.6.3 The `Speculation-Rules`[p1132] header §[p11 32]

The `Speculation-Rules` HTTP response header allows the developer to request that the user agent fetch and apply a given speculation rule set[p1120] to the current Document[p135]. It is a structured header whose value must be a list of strings that are all valid URL strings.

To **process the `Speculation-Rules` header** given a Document[p135] *document* and a response *response*:

1. Let *parsedList* be the result of getting a structured field value given `Speculation-Rules`[p1132] and "list" from *response*'s header list.
2. If *parsedList* is null, then return.
3. For each *item* of *parsedList*:
    1. If *item* is not a string, then continue.
    2. Let *url* be the result of URL parsing *item* with *document*'s document base URL[p101].
    3. If *url* is failure, then continue.
    4. In parallel[p44]:
        1. Optionally, wait for an implementation-defined amount of time.

            Note

            This allows the implementation to prioritize other work ahead of loading speculation rules, as especially during Document[p135] creation and header processing, there are often many more important things going on.

        2. Queue a global task[p1195] on the speculation rules task source[p1132] given *document*'s relevant global object[p1151] to perform the following steps:
            1. Let *request* be a new request whose URL is *url*, destination is "speculationrules", and mode is "cors".
            2. Fetch *request* with the following *processResponseConsumeBody* steps given response *response* and null, failure, or a byte sequence *bodyBytes*:
                1. If *bodyBytes* is null or failure, then abort these steps.
                2. If *response*'s status is not an ok status, then abort these steps.
                3. If the result of extracting a MIME type from *response*'s header list does not have an essence of "application/speculationrules+json[p1551]", then abort these steps.
                4. Let *bodyText* be the result of UTF-8 decoding *bodyBytes*.
                5. Let *ruleSet* be the result of parsing a speculation rule set string[p1122] given *bodyText*, *document*, and *response*'s URL. If this throws an exception, then abort these steps.
                6. Append *ruleSet* to *document*'s speculation rule sets[p1128].
                7. Consider speculative loads[p1128] for *document*.

7.6.4 The `Sec-Speculation-Tags`[p1133] header §[p11 33]

The `Sec-Speculation-Tags` HTTP request header specifies the web developer-provided tags associated with the speculative navigation request. It can also be used to distinguish speculative navigation requests from speculative subresource requests, since `Sec-Purpose` can be sent by both categories of requests.

The header is a structured header whose value must be a list. The list can contain either token or string values. String values represent developer-provided tags, whereas token values represent predefined tags. As of now, the only predefined tag is `null`, which indicates a speculative navigation request with no developer-defined tag.

7.6.5 Security considerations §[p11 33]

7.6.5.1 Cross-site requests §[p11 33]

Speculative loads can be initiated by web pages to cross-site destinations. However, because such cross-site speculative loads are

always done without credentials, as explained below[p1135], ambient authority is limited to requests that are already possible via other mechanisms on the platform.

The `Speculation-Rules[p1132]` header can also be used to issue requests, for JSON documents whose body will be parsed as a speculation rule set string[p1122]. However, they use the "same-origin" credentials mode, the "cors" mode, and responses which do not use the application/speculationrules+json[p1551] MIME type essence are ignored, so they are not useful in mounting attacks.

7.6.5.2 Injected content §[p11 34]

Because links in a document can be selected for speculative loading via document rule predicates[p1120], developers need to be cautious if such links might contain user-generated markup. For example, if the href[p314] of a link can be entered by one user and displayed to all other users, a malicious user might choose a value like "/logout", causing other users' browsers to automatically log out of the site when that link is speculatively loaded. Using a document rule selector predicate[p1121] to exclude such potentially-dangerous links, or using a document rule URL pattern predicate[p1121] to allowlist known-safe links, are useful techniques in this regard.

As with all uses of the script[p685] element, developers need to be cautious about inserting user-provided content into <script type=speculationrules>'s child text content. In particular, the insertion of an unescaped closing </script> tag could be used to break out of the script[p685] element context and inject attacker-controlled markup.

The <script type=speculationrules> feature causes activity in response to content found in the document, so it is worth considering the options open to an attacker able to inject unescaped HTML. Such an attacker is already able to inject JavaScript or iframe[p404] elements. Speculative loads are generally less dangerous than arbitrary script execution. However, the use of document rule predicates[p1120] could be used to speculatively load links in the document, and the existence of those loads could provide a vector for exfiltrating information about those links. Defense-in-depth against this possibility is provided by Content Security Policy. In particular, the script-src directive can be used to restrict the parsing of speculation rules script[p685] elements, and the default-src directive applies to navigational prefetch requests arising from such speculation rules. Additional defense is provided by the requirement that speculative loads are only performed to potentially-trustworthy URLs, so an on-path attacker would only have access to metadata and traffic analysis, and could not see the URLs directly. [CSP][p1580]

It's generally not expected that user-generated content will be added as arbitrary response headers: server operators are already going to encounter significant trouble if this is possible. It is therefore unlikely that the `Speculation-Rules[p1132]` header meaningfully expands the XSS attack surface. For this reason, Content Security Policy does not apply to the loading of rule sets via that header.

7.6.5.3 IP anonymization §[p11 34]

This standard allows developers to request that navigational prefetches are performed using IP anonymization technology provided by the user agent. The details of this anonymization are not specified, but some general security principles apply.

To the extent IP anonymization is implemented using a proxy service, it is advisable to minimize the information available to the service operator and other entities on the network path. This likely involves, at a minimum, the use of TLS for the connection.

Site operators need to be aware that, similar to virtual private network (VPN) technology, the client IP address seen by the HTTP server might not exactly correspond to the user's actual network provider or location, and a traffic for multiple distinct subscribers could originate from a single client IP address. This can affect site operators' security and abuse prevention measures. IP anonymization measures might make an effort to use an egress IP address which has a similar geolocation or is located in the same jurisdiction as the user, but any such behavior is particular to the user agent and not guaranteed.

7.6.6 Privacy considerations §[p11 34]

7.6.6.1 Heuristics and optionality §[p11 34]

The consider speculative loads[p1128] algorithm contains a crucial "may" step, which encourages user agents to start referrer-initiated navigational prefetches based on a combination of the speculation rule eagerness[p1121] and other features of the user's environment. Because it can be observable to the document whether speculative loads are performed, user agents must take care to protect privacy when making such decisions—for instance by only using information which is already available to the origin. If these heuristics depend on any persistent state, that state must be erased whenever the user erases other site data. If the user agent automatically clears other site data from time to time, it must erase such persistent state at the same time.

Note

The use of origin[p939] instead of site[p940] here is intentional. Although same-site origins are generally allowed to coordinate if they wish, the web's security model is premised on preventing origins from accessing the data of other origins, even same-site ones. Thus, the user agent needs to be sure not to leak such data unintentionally across origins, not just across sites.

Examples of inputs which would be already known to the document:

- author-supplied eagerness[p1121]
- order of appearance in the document
- whether a link is in the viewport
- whether the cursor is near a link
- rendered size of a link

Examples of persistent data related to the origin (which the origin could have gathered itself) but which must be erased according to user intent:

- whether the user has clicked this or similar links on this document or other documents on the same origin

Examples of device information which might be valuable in deciding whether speculative loading is appropriate, but which needs to be considered as part of the user agent's overall privacy posture because it can make the user more identifiable across origins:

- coarse device class (CPU, memory)
- coarse battery level
- whether the network connection is known to be metered
- any user-toggleable settings, such as a speculative loading toggle, a battery-saver toggle, or a data-saver toggle

7.6.6.2 State partitioning §[p1135]

The start a referrer-initiated navigational prefetch algorithm is designed to ensure that the HTTP requests that it issues behave consistently with how user agents partition credentials according to storage keys. This property is maintained even for cross-partition prefetches, as follows.

If a future navigation using a prefetched response would load a document in the same partition, then at prefetch time, the partitioned credentials can be sent, as they can with subresource requests and scripted fetches. If such a future navigation would instead load a document in another partition, it would be inconsistent with the partitioning scheme to use partitioned credentials for the destination partition (since this would cross the boundary between partitions without a top-level navigation) and also inconsistent to use partitioned credentials within the originating partition (since this would result in the user seeing a document with different state than a non-prefetched navigation). Instead, a third, initially empty, partition is used for such requests. These requests therefore send along no credentials from either partition. However, the resulting prefetched response body constructed using this initially-empty partition can only be used if, at activation time, the destination partition contains no credentials.

This is somewhat similar to the behavior of only sending such prefetch requests if the destination partition is known ahead of time to not contain credentials. However, to avoid such behavior being used a way of probing for the presence of credentials, instead such prefetch requests are always completed, and in the case of conflicting credentials, their results are not used.

Redirects are possible between these two types of requests. A redirect from a same- to cross-partition URL could contain information derived from partitioned credentials in the originating partition; however, this is equivalent to the originating document fetching the same-partition URL itself and then issuing a request for the cross-partition URL. A redirect from a cross- to same-origin URL could carry credentials from the isolated partition, but since this partition has no prior state this does not enable tracking based on the user's prior browsing activity on that site, and the document could construct the same state by issuing uncredentialed requests itself.

7.6.6.3 Identity joining §[p1135]

Speculative loads provide a mechanism through which HTTP requests for later top-level navigation can be made without a user gesture. It is natural to ask whether it is possible for two coordinating sites to connect user identities.

Since existing credentials for the destination site are not sent (as explained in the previous section), that site is limited in its ability to identify the user before navigation in a similar way to if the referrer site had simply used `fetch()` to make an uncredentialed request. Upon navigation, this becomes similar to ordinary navigation (e.g., by clicking a link that was not speculatively loaded).

To the extent that user agents attempt to mitigate identity joining for ordinary fetches and navigations, they can apply similar mitigations to speculatively-loaded navigations.

7.7 The `X-Frame-Options` header

The **`X-Frame-Options`** HTTP response header is a way of controlling whether and how a Document may be loaded inside of a child navigable. For sites using CSP, the `frame-ancestors` directive 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] [RFC7034]

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` *header are used in the same response, then* `X-Frame-Options` *is ignored.*

For web developers and conformance checkers, its value ABNF is:

```
X-Frame-Options = "DENY" / "SAMEORIGIN"
```

To **check a navigation response's adherence to `X-Frame-Options`**, given a response *response*, a navigable *navigable*, a CSP list *cspList*, and an origin *destinationOrigin*:

1. If *navigable* is not a child navigable, then return true.
2. For each *policy* of *cspList*:
    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` 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` *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`":

    1. Let *containerDocument* be *navigable*'s container document[p1038].

    2. While *containerDocument* is not null:

        1. If *containerDocument*'s origin is not same origin[p940] with *destinationOrigin*, then return false.

        2. Set *containerDocument* to *containerDocument*'s container document[p1039].

10. Return true.

Note

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:

`X-Frame-Options`[p1136]	Valid	Result
`DENY`	✔	embedding disallowed
`SAMEORIGIN`	✔	same-origin embedding allowed
`INVALID`	✘	embedding allowed
`ALLOWALL`	✘	embedding allowed
`ALLOW-FROM=https://example.com/`	✘	embedding allowed (from anywhere)

Example

The following table illustrates how various non-conformant cases involving multiple values are processed:

`X-Frame-Options`[p1136]	Result
`SAMEORIGIN, SAMEORIGIN`	same-origin embedding allowed
`SAMEORIGIN, DENY`	embedding disallowed
`SAMEORIGIN,`	embedding disallowed
`SAMEORIGIN, ALLOWALL`	embedding disallowed
`SAMEORIGIN, INVALID`	embedding disallowed
`ALLOWALL, INVALID`	embedding disallowed
`ALLOWALL,`	embedding disallowed
`INVALID, INVALID`	embedding allowed

The same results are obtained whether the values are delivered in a single header whose value is comma-delimited, or in multiple headers.

7.8 The `Refresh`[p1137] header §[p11 37]

The `Refresh` HTTP response header is the HTTP-equivalent to a `meta`[p196] element with an `http-equiv`[p202] attribute in the Refresh state[p204]. It takes the same value[p205] and works largely the same. Its processing model is detailed in create and initialize a `Document` object[p1106].

7.9 Browser user interface considerations §[p11 37]

Browser user agents should provide the ability to navigate[p1063], reload[p1077], and stop loading[p1118] any top-level traversable[p1037] in their top-level traversable set[p1037].

Example

For example, via a location bar and reload/stop button UI.

Browser user agents should provide the ability to traverse by a delta[p1077] any top-level traversable[p1037] in their top-level traversable set[p1037].

Example

For example, via back and forward buttons, possibly including long-press abilities to change the delta.

It is suggested that such user agents allow traversal by deltas greater than one, to avoid letting a page "trap" the user by stuffing the session history with spurious entries. (For example, via repeated calls to `history.pushState()`[p989] or fragment navigations[p1071].)

Note

Some user agents have heuristics for translating a single "back" or "forward" button press into a larger delta, specifically to overcome such abuses. We are contemplating specifying these heuristics in issue #7832.

Browser user agents should offer users the ability to create a fresh top-level traversable[p1038], given a user-provided or user agent-determined initial URL.

Example

For example, via a "new tab" or "new window" button.

Browser user agents should offer users the ability to arbitrarily close[p1042] any top-level traversable[p1037] in their top-level traversable set[p1037].

Example

For example, by clicking a "close tab" button.

Browser user agents may provide ways for the user to explicitly cause any navigable[p1035] (not just a top-level traversable[p1037]) to navigate[p1063], reload[p1077], or stop loading[p1118].

Example

For example, via a context menu.

Browser user agents may provide the ability for users to destroy a top-level traversable[p1042].

Example

For example, by force-closing a window containing one or more such top-level traversables[p1037].

When a user requests a reload[p1077] of a navigable[p1035] whose active session history entry[p1035]'s document state[p1053]'s resource[p1054] is a POST resource[p1055], the user agent should prompt the user to confirm the operation first, since otherwise transactions (e.g., purchases or database modifications) could be repeated.

When a user requests a reload[p1077] of a navigable[p1035], user agents may provide a mechanism for ignoring any caches when reloading.

All calls to navigate[p1063] initiated by the mechanisms mentioned above must have the *userInvolvement*[p1063] argument set to "`browser UI`[p1062]".

All calls to reload[p1077] initiated by the mechanisms mentioned above must have the *userInvolvement*[p1077] argument set to "`browser UI`[p1062]".

All calls to traverse the history by a delta[p1077] initiated by the mechanisms mentioned above must not pass a value for the *sourceDocument*[p1077] argument.

The above recommendations, and the data structures in this specification, are not meant to place restrictions on how user agents represent the session history to the user.

For example, although a top-level traversable[p1037]'s session history entries[p1037] are stored and maintained as a list, and the user agent is recommended to give an interface for traversing that list by a delta[p1077], a novel user agent could instead or in addition present a tree-like view, with each page having multiple "forward" pages that the user can choose between.

Similarly, although session history for all descendant navigables[p1035] is stored in their traversable navigable[p1037], user agents could present the user with a more nuanced per-navigable[p1035] view of the session history.

Browser user agents may use a top-level browsing context[p1049]'s is popup[p1046] boolean for the following purposes:

- Deciding whether or not to provide a minimal web browser user interface for the corresponding top-level traversable[p1037].
- Performing the optional steps in set up browsing context features.

In both cases user agents might additionally incorporate user preferences, or present a choice as to whether to go down the popup route.

User agents that provide a minimal user interface for such popups are encouraged to not hide the browser's location bar.

8 Web application APIs §p11 40

8.1 Scripting §p11 40

8.1.1 Introduction §p11 40

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`p685 elements.
- Navigating to `javascript:` URLsp1068.
- Event handlers, whether registered through the DOM using `addEventListener()`, by explicit event handler content attributesp1207, by event handler IDL attributesp1207, or otherwise.
- Processing of technologies like SVG that have their own scripting features.

8.1.2 Agents and agent clusters §p11 40

8.1.2.1 Integration with the JavaScript agent formalism §p11 40

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/realmsp1145 that can synchronously access each other, and thus needs to run in a single execution thread.

Two `Window`p965 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-domainp940; see IsPlatformObjectSameOriginp963.

The following types of agents exist on the web platform:

Similar-origin window agent

Contains various `Window`p965 objects which can potentially reach each other, either directly or by using `document.domain`p942.

If the encompassing agent cluster's is origin-keyedp1141 is true, then all the `Window`p965 objects will be same originp940, can reach each other directly, and `document.domain`p942 will no-op.

Note

Two `Window`p965 objects that are same originp940 can be in different similar-origin window agentsp1140, for instance if they are each in their own browsing context groupp1050.

Dedicated worker agent

Contains a single `DedicatedWorkerGlobalScope`p1324.

Shared worker agent

Contains a single `SharedWorkerGlobalScope`p1324.

Service worker agent

Contains a single `ServiceWorkerGlobalScope`.

Worklet agent

Contains a single `WorkletGlobalScope`p1342 object.

Note

Although a given worklet can have multiple realms, each such realm needs its own agent, as each realm can be executing code independently and at the same time as the others.

Only shared[p1140] and dedicated worker agents[p1140] 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 [[CanBlock]] is *canBlock*, [[Signifier]] is *signifier*, [[CandidateExecution]] is *candidateExecution*, and [[IsLockFree1]], [[IsLockFree2]], and [[LittleEndian]] are set at the implementation's discretion.
4. Set *agent*'s event loop[p1193] to a new event loop[p1193].
5. Return *agent*.

For a realm *realm*, the agent whose [[Signifier]] is *realm*.[[AgentSignifier]] is **the realm's agent**.

The **relevant agent** for a platform object *platformObject* is *platformObject*'s relevant realm[p1151]'s agent[p1141].

Note

The agent equivalent of the current realm is the surrounding agent.

8.1.2.2 Integration with the JavaScript agent cluster formalism §[p11 41]

JavaScript also defines the concept of an agent cluster, which this standard maps to the web platform by placing agents appropriately when they are created using the obtain a similar-origin window agent[p1141] or obtain a worker/worklet agent[p1142] algorithms.

The agent cluster concept is crucial for defining the JavaScript memory model, and in particular among which agents the backing data of `SharedArrayBuffer` objects can be shared.

Note

Conceptually, the agent cluster concept is an architecture-independent, idealized "process boundary" that groups together multiple "threads" (agents). The agent clusters defined by the specification are generally more restrictive than the actual process boundaries implemented in user agents. By enforcing these idealized divisions at the specification level, we ensure that web developers see interoperable behavior with regard to shared memory, even in the face of varying and changing user agent process models.

An agent cluster has an associated **cross-origin isolation mode**, which is a cross-origin isolation mode[p1050]. It is initially "none[p1050]".

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[p1140].

An **agent cluster key** is a site[p940] or tuple origin[p939]. Without web developer action to achieve origin-keyed agent clusters[p944], it will be a site[p940].

Note

An equivalent formulation is that an agent cluster key[p1141] can be a scheme-and-host[p940] or an origin[p939].

To **obtain a similar-origin window agent**, given an origin[p939] *origin*, a browsing context group[p1050] *group*, and a boolean *requestsOAC*, run these steps:

1. Let *site* be the result of obtaining a site[p940] with *origin*.
2. Let *key* be *site*.

3. If *group*'s cross-origin isolation mode[p1050] is not "none[p1050]", then set *key* to *origin*.
4. Otherwise, if *group*'s historical agent cluster key map[p1050][*origin*] exists, then set *key* to *group*'s historical agent cluster key map[p1050][*origin*].
5. Otherwise:
    1. If *requestsOAC* is true, then set *key* to *origin*.
    2. Set *group*'s historical agent cluster key map[p1050][*origin*] to *key*.
6. If *group*'s agent cluster map[p1050][*key*] does not exist:
    1. Let *agentCluster* be a new agent cluster.
    2. Set *agentCluster*'s cross-origin isolation mode[p1141] to *group*'s cross-origin isolation mode[p1050].
    3. If *key* is an origin[p939]:
        1. Assert: *key* is *origin*.
        2. Set *agentCluster*'s is origin-keyed[p1141] to true.
    4. Add the result of creating an agent[p1141], given false, to *agentCluster*.
    5. Set *group*'s agent cluster map[p1050][*key*] to *agentCluster*.
7. Return the single similar-origin window agent[p1140] contained in *group*'s agent cluster map[p1050][*key*].

Note

This means that there is only one similar-origin window agent[p1140] per browsing context agent cluster. (However, dedicated worker[p1140] and worklet agents[p1140] 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[p1144] or null *outside settings*, a boolean *isTopLevel*, and a boolean *canBlock*, run these steps:

1. Let *agentCluster* be null.
2. If *isTopLevel* is true:
    1. Set *agentCluster* to a new agent cluster.
    2. Set *agentCluster*'s is origin-keyed[p1141] 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`[p944] exposes the origin-keyedness for windows).
3. Otherwise:
    1. Assert: *outside settings* is not null.
    2. Let *ownerAgent* be *outside settings*'s realm[p1145]'s agent[p1141].
    3. Set *agentCluster* to the agent cluster which contains *ownerAgent*.
4. Let *agent* be the result of creating an agent[p1141] given *canBlock*.
5. Add *agent* to *agentCluster*.
6. Return *agent*.

To **obtain a dedicated/shared worker agent**, given an environment settings object[p1144] *outside settings* and a boolean *isShared*, return the result of obtaining a worker/worklet agent[p1142] given *outside settings*, *isShared*, and true.

To **obtain a worklet agent**, given an environment settings object[p1144] *outside settings*, return the result of obtaining a worker/worklet

agent[p1142] given *outside settings*, false, and false.

To **obtain a service worker agent**, return the result of obtaining a worker/worklet agent[p1142] given null, true, and false.

¶[p11 43] **Example**

The following pairs of global objects are each within the same agent cluster, and thus can use `SharedArrayBuffer` instances to share memory with each other:

- A `Window`[p965] object and a dedicated worker that it created.
- A worker (of any type) and a dedicated worker it created.
- A `Window`[p965] object *A* and the `Window`[p965] object of an `iframe`[p404] element that *A* created that could be same origin-domain[p940] with *A*.
- A `Window`[p965] object and a same origin-domain[p940] `Window`[p965] object that opened it.
- A `Window`[p965] object and a worklet that it created.

The following pairs of global objects are *not* within the same agent cluster, and thus cannot share memory:

- A `Window`[p965] object and a shared worker it created.
- A worker (of any type) and a shared worker it created.
- A `Window`[p965] object and a service worker it created.
- A `Window`[p965] object *A* and the `Window`[p965] object of an `iframe`[p404] element that *A* created that cannot be same origin-domain[p940] with *A*.
- Any two `Window`[p965] objects with no opener or ancestor relationship. This holds even if the two `Window`[p965] objects are same origin[p940].

8.1.3 Realms and their counterparts §[p11 43]

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[p1145]; 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`[p965] or `WorkerGlobalScope`[p1322] interfaces. When the values have utility across multiple realms, we use the environment settings object[p1144] 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[p1063]). These values are tracked in the environment[p1143] concept.

8.1.3.1 Environments §[p11 43]

An **environment** is an object that identifies the settings of a current or potential execution environment (i.e., a navigation params[p1061]'s reserved environment[p1061] or a request's reserved client). An environment[p1143] has the following fields:

An *id*

An opaque string that uniquely identifies this environment[p1143].

A *creation URL*

A URL that represents the location of the resource with which this environment[p1143] is associated.

Note

In the case of a Window[p965] environment settings object[p1144], this URL might be distinct from its global object[p1145]'s associated Document[p966]'s URL, due to mechanisms such as history.pushState()[p989] which modify the latter.

A *top-level creation URL*

Null or a URL that represents the creation URL[p1143] of the "top-level" environment[p1143]. It is null for workers and worklets.

A *top-level origin*

A for now implementation-defined value, null, or an origin[p939]. 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[p1143]'s origin[p1144]. For a dedicated worker or worklet it is the top-level origin[p1144] 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[p1144]'s origin when sandboxing, workers, and worklets are involved.

A *target browsing context*

Null or a target browsing context[p1046] for a navigation request.

An *active service worker*

Null or a service worker that controls the environment[p1143].

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[p1143] as input.

Note

The environment discarding steps[p1144] 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 §[p1144]

An **environment settings object** is an environment[p1143] that additionally specifies algorithms for:

A *realm execution context*

A JavaScript execution context shared by all scripts[p685] that use this settings object, i.e., all scripts in a given realm. When we run a classic script[p1165] or run a module script[p1165], 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 Source Text Module Record's Evaluate knows which realm to use.)

A *module map*

A module map[p1189] that is used when importing JavaScript modules.

An *API base URL*

A URL used by APIs called by scripts that use this environment settings object[p1144] to parse URLs[p101].

An *origin*

An origin[p939] used in security checks.

A *has cross-site ancestor*

A boolean used in security checks.

A *policy container*

A policy container[p960] containing policies used for security checks.

A *cross-origin isolated capability*

A boolean representing whether scripts that use this environment settings object[p1144] are allowed to use APIs that require cross-origin isolation.

A *time origin*

A number used as the baseline for performance-related timestamps. [HRT]p1582

An environment settings objectp1144's **responsible event loop** is its global objectp1145's relevant agentp1141's event loopp1193.

8.1.3.3 Realms, settings objects, and global objects §p1145

A **global object** is a JavaScript object that is the [[GlobalObject]] field of a realm.

Note

In this specification, all realms are createdp1145 with global objectsp1145 that are either `Window`p965, `WorkerGlobalScope`p1322, *or* `WorkletGlobalScope`p1342 *objects.*

A global objectp1145 has an **in error reporting mode** boolean, which is initially false.

A global objectp1145 has an **outstanding rejected promises weak set**, a set of `Promise` objects, initially empty. This set must not create strong references to any of its members, and implementations are free to limit its size in an implementation-defined manner, e.g., by removing old entries from it when new ones are added.

A global objectp1145 has an **about-to-be-notified rejected promises list**, a list of `Promise` objects, initially empty.

A global objectp1145 has an **import map**, initially an empty import mapp1177.

Note

For now, only `Window`p965 *global objectsp1145 have their import mapp1145 modified from the initial empty one. The import mapp1145 is only accessed for the resolution of a root module scriptp1153.*

A global objectp1145 has a **resolved module set**, a set of specifier resolution recordsp1177, initially empty.

Note

The resolved module setp1145 ensures that module specifier resolution returns the same result when called multiple times with the same (referrer, specifier) pair. It does that by ensuring that import mapp1177 rules that impact the specifier in its referrer's scope cannot be defined after its initial resolution. For now, only `Window`p965 *global objectsp1145 have their module set data structures modified from the initial empty one.*

There is always a 1-to-1-to-1 mapping between realms, global objectsp1145, and environment settings objectsp1144:

- A realm has a [[HostDefined]] field, which contains **the realm's settings object**.
- A realm has a [[GlobalObject]] field, which contains **the realm's global object**.
- Each global objectp1145 in this specification is created during the creationp1145 of a corresponding realm, known as **the global object's realm**.
- Each global objectp1145 in this specification is created alongside a corresponding environment settings objectp1144, known as its relevant settings objectp1151.
- An environment settings objectp1144's realm execution contextp1144's Realm component is **the environment settings object's realm**.
- An environment settings objectp1144's realmp1145 then has a [[GlobalObject]] field, which contains **the environment settings object's global object**.

To **create a new realm** in an agent *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.

Note

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. If *agent*'s agent cluster's cross-origin isolation mode[p1141] is "none[p1050]":
    1. Let *global* be *realm*'s global object[p1145].
    2. Let *status* be ! *global*.[[Delete]]("`SharedArrayBuffer`").
    3. Assert: *status* is true.

Note

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`.

6. Return *realm execution context*.

When defining algorithm steps throughout this specification, it is often important to indicate what realm is to be used—or, equivalently, what global object[p1145] or environment settings object[p1144] 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 realm.)

Relevant

Every platform object has a relevant realm[p1151], which is roughly the realm in which it was created. When writing algorithms, the most prominent platform object whose relevant realm[p1151] might be important is the **this** value of the currently-running function object. In some cases, there can be other important relevant realms[p1151], such as those of any arguments.

Note how the entry[p1146], incumbent[p1146], and current[p1146] concepts are usable without qualification, whereas the relevant[p1146] concept must be applied to a particular platform object.

⚠Warning!

The incumbent[p1146] and entry[p1146] 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[p1146], and issue #1431 for entry[p1146].

In general, web platform specifications should use the relevant[p1146] concept, applied to the object being operated on (usually the **this** value of the current method). This mismatches the JavaScript specification, where current[p1146] is generally used as the default (e.g., in determining the 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.

Example

Consider the following pages, with `a.html` being loaded in a browser window, `b.html` being loaded in an `iframe`[p404] as shown, and `c.html` and `d.html` omitted (they can simply be empty documents):

```
<!-- 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[p1046], and thus its own realm, global object[p1145], and environment settings object[p1144].

When the print()[p1260] method is called in response to pressing the button in `a.html`:

- The entry realm[p1148] is that of `a.html`.
- The incumbent realm[p1149] is that of `b.html`.
- The current realm is that of `c.html` (since it is the print()[p1260] method from `c.html` whose code is running).
- The relevant realm[p1151] of the object on which the print()[p1260] method is being called is that of `d.html`.

Example

One reason why the relevant[p1146] concept is generally a better default choice than the current[p1146] 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[p1151] for the Navigator[p1261] object on which it is invoked. This has the following impact: [BATTERY][p1579]

```
<!-- outer.html -->
<!DOCTYPE html>
<html lang="en">
<title>Relevant realm demo: outer page</title>
<script>
  function doTest() {
    const promise = navigator.getBattery.call(frames[0].navigator);

    console.log(promise instanceof Promise);           // logs false
    console.log(promise instanceof frames[0].Promise); // logs true

    frames[0].hello();
  }
</script>
<iframe src="inner.html" onload="doTest()"></iframe>
```

```
<!-- inner.html -->
<!DOCTYPE html>
<html lang="en">
<title>Relevant realm demo: inner page</title>
<script>
  function hello() {
    const promise = navigator.getBattery();

    console.log(promise instanceof Promise);        // logs true
    console.log(promise instanceof parent.Promise); // logs false
  }
</script>
```

If the algorithm for the `getBattery()` method had instead used the current realm, all the results would be reversed. That is, after the first call to `getBattery()` in `outer.html`, the `Navigator` object in `inner.html` would be permanently storing a `Promise` object created in `outer.html`'s 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[p1149].

   Note

   This will be the same as the topmost script-having execution context[p1149] inside the corresponding invocation of prepare to run a callback[p1148].

2. If *context* is not null, decrement *context*'s skip-when-determining-incumbent counter[p1148].
3. Assert: the topmost entry of the backup incumbent settings object stack[p1148] is *settings*.
4. Remove *settings* from the backup incumbent settings object stack[p1148].

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[p1149].
2. If *context* is null, or if *context*'s skip-when-determining-incumbent counter[p1148] is greater than zero:
   1. Assert: the backup incumbent settings object stack[p1148] is not empty.

      Note

      This assert would fail if you try to obtain the incumbent settings object[p1149] from inside an algorithm that was triggered neither by calling scripts[p1165] nor by Web IDL invoking a callback. For example, it would trigger if you tried to obtain the incumbent settings object[p1149] inside an algorithm that ran periodically as part of the event loop[p1193], with no involvement of author code. In such cases the incumbent[p1146] concept cannot be used.

   2. Return the topmost entry of the backup incumbent settings object stack[p1148].
3. Return *context*'s Realm component's settings object[p1145].

Then, the **incumbent realm** is the realm[p1145] of the incumbent settings object[p1149].

Similarly, the **incumbent global object** is the global object[p1145] of the incumbent settings object[p1149].

The following series of examples is intended to make it clear how all of the different mechanisms contribute to the definition of the incumbent[p1148] concept:

¶ p1149 Example

Consider the following starter example:

```
<!DOCTYPE html>
<iframe></iframe>
<script>
  frames[0].postMessage("some data", "*");
</script>
```

There are two interesting environment settings objects[p1144] here: that of `window`, and that of `frames[0]`. Our concern is: what is the incumbent settings object[p1149] at the time that the algorithm for `postMessage()`[p1296] 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[p1295] use the incumbent settings object[p1149] to determine the `source`[p1284] property of the resulting `MessageEvent`[p1283], 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[p1151].

When the window post message steps[p1295] look up the incumbent settings object[p1149], the topmost script-having execution context[p1149] will be that corresponding to the `script`[p685] element: it was pushed onto the JavaScript execution context stack as part of ScriptEvaluation during the run a classic script[p1165] algorithm. Since there are no Web IDL callback invocations involved, the

context's skip-when-determining-incumbent counter[p1148] is zero, so it is used to determine the incumbent settings object[p1149]; the result is the environment settings object[p1144] of `window`.

(Note how the environment settings object[p1144] of `frames[0]` is the relevant settings object[p1151] of this at the time the `postMessage()`[p1296] method is called, and thus is involved in determining the *target* of the message. Whereas the incumbent is used to determine the *source*.)

¶[p1150] **Example**

Consider the following more complicated example:

```
<!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()`[p1253]. But, the answer should be the same: invoking algorithms asynchronously should not change the incumbent[p1146] concept.

This time, the result involves more complicated mechanisms:

When `bound` is converted to a Web IDL callback type, the incumbent settings object[p1149] 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[p1194] posted by `setTimeout()`[p1253] 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[p1148] algorithm. This pushes the stored callback context onto the backup incumbent settings object stack[p1148]. At this time (inside the timer task) there is no author code on the stack, so the topmost script-having execution context[p1149] is null, and nothing gets its skip-when-determining-incumbent counter[p1148] incremented.

Invoking the callback then calls `bound`, which in turn calls the `postMessage()`[p1296] method of `frames[0]`. When the `postMessage()`[p1296] algorithm looks up the incumbent settings object[p1149], 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[p1149] will be null: the JavaScript execution context stack only contains an execution context corresponding to `postMessage()`[p1296], with no ScriptEvaluation context or similar below it.

This is where we fall back to the backup incumbent settings object stack[p1148]. As noted above, it will contain as its topmost entry the relevant settings object[p1151] of `window`. So that is what is used as the incumbent settings object[p1149] while executing the `postMessage()`[p1296] algorithm.

¶[p1150] **Example**

Consider this final, even more convoluted example:

```
<!-- a.html -->
<!DOCTYPE html>
<button>click me</button>
<iframe></iframe>
<script>
const bound = frames[0].location.assign.bind(frames[0].location, "https://example.com/");
document.querySelector("button").addEventListener("click", bound);
</script>
```

```
<!-- b.html -->
<!DOCTYPE html>
<iframe src="a.html"></iframe>
<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[p1144] in play: that of `a.html`, and that of `b.html`. When the `location.assign()`[p985] method triggers the `Location`-object navigate[p981] algorithm, what will be the incumbent settings object[p1149]? As before, it should intuitively be that of `a.html`: the `click` listener was originally scheduled by `a.html`, so even if something involving `b.html` causes the listener to fire, the incumbent[p1146] 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[p1149] is that corresponding to `a.html`, which is stored as the callback's callback context.

When the `click()`[p871] method is called inside `b.html`, it dispatches a `click` event on the button that is inside `a.html`. This time, when the prepare to run a callback[p1148] algorithm executes as part of event dispatch, there *is* author code on the stack; the topmost script-having execution context[p1149] is that of the `onLoad` function, whose skip-when-determining-incumbent counter[p1148] gets incremented. Additionally, `a.html`'s environment settings object[p1144] (stored as the `EventHandler`[p1211]'s callback context) is pushed onto the backup incumbent settings object stack[p1148].

Now, when the `Location`-object navigate[p981] algorithm looks up the incumbent settings object[p1149], the topmost script-having execution context[p1149] is still that of the `onLoad` function (due to the fact we are using a bound function as the callback). Its skip-when-determining-incumbent counter[p1148] value is one, however, so we fall back to the backup incumbent settings object stack[p1148]. This gives us the environment settings object[p1144] of `a.html`, as expected.

Note that this means that even though it is the `iframe`[p404] inside `a.html` that navigates, it is `a.html` itself that is used as the source `Document`[p135], 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[p1146] or relevant[p1146] as appropriate.

8.1.3.3.3 Current §[p11 51]

The JavaScript specification defines the current realm, also known as the "current Realm Record". [JAVASCRIPT][p1583]

Then, the **current settings object** is the environment settings object[p1145] of the current realm.

Similarly, the **current global object** is the global object[p1145] of the current realm.

8.1.3.3.4 Relevant §[p11 51]

The **relevant realm** for a platform object is the value of its [[Realm]] field.

Then, the **relevant settings object** for a platform object *o* is the environment settings object[p1145] of the relevant realm[p1151] for *o*.

Similarly, the **relevant global object** for a platform object *o* is the global object[p1145] of the relevant realm[p1151] for *o*.

8.1.3.4 Enabling and disabling scripting §[p11 51]

Scripting is enabled for an environment settings object[p1144] *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[p1144].)
- Either *settings*'s global object[p1145] is not a `Window`[p965] object, or *settings*'s global object[p1145]'s associated

Document[p966]'s active sandboxing flag set[p959] does not have its sandboxed scripts browsing context flag[p957] set.

- The result of WebDriver BiDi scripting is enabled with *settings* is true.

Scripting is disabled for an environment settings object[p1144] when scripting is not enabled[p1151] for it, i.e., when any of the above conditions are false.

Scripting is disabled for a platform object *object* if any of the following are true:

- Scripting is disabled[p1152] for *object*'s relevant settings object[p1151].
- The *object* implements Node, and *object*'s node document's browsing context[p1046] is null.
- The *object* implements Window[p965] and *object*'s associated Document[p966]'s browsing context[p1046] is null.

Scripting is enabled for a platform object *object*, when *object*'s scripting is not disabled[p1152].

8.1.3.5 Secure contexts §[p11 52]

An environment[p1143] *environment* is a **secure context** if the following algorithm returns true:

1. If *environment* is an environment settings object[p1144]:
    1. Let *global* be *environment*'s global object[p1145].
    2. If *global* is a WorkerGlobalScope[p1322]:
        1. If *global*'s owner set[p1322][0]'s relevant settings object[p1151] is a secure context[p1152], then return true.

            Note

            We only need to check the 0th item since they will necessarily all be consistent.

        2. Return false.
    3. If *global* is a WorkletGlobalScope[p1342], 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[p1144] is "`Potentially Trustworthy`", then return true.
3. Return false.

An environment[p1143] is a **non-secure context** if it is not a secure context[p1152].

8.1.4 Script processing model §[p11 52]

8.1.4.1 Scripts §[p11 52]

A **script** is one of two possible structs (namely, a classic script[p1153] or a module script[p1153]). All scripts have:

A *settings object*

An environment settings object[p1144], containing various settings that are shared with other scripts[p1152] in the same context.

A *record*

One of the following:

- a script record, for classic scripts[p1153];
- a Source Text Module Record, for JavaScript module scripts[p1153];

- a Synthetic Module Record, for CSS module scripts[p1153], JSON module scripts[p1153], and text module scripts[p1153];
- a WebAssembly Module Record, for WebAssembly module scripts[p1153]; or
- null, representing a parsing failure.

A *parse error*

A JavaScript value, which has meaning only if the record[p1152] 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[p1161], and is not to be used directly. Instead, consult the error to rethrow[p1153] 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[p1165] the script.

Note

This could be the script's parse error[p1153], but in the case of a module script[p1153] it could instead be the parse error[p1153] from one of its dependencies, or an error from resolve a module specifier[p1171].

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

Null or a script fetch options[p1154], containing various options related to fetching this script or module scripts[p1153] that it imports.

A *base URL*

Null or a base URL used for resolving module specifiers[p1171]. When non-null, this will either be the URL from which the script was obtained, for external scripts, or the document base URL[p101] of the containing document, for inline scripts.

A **classic script** is a type of script[p1152] 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[p1152]. It has no additional items.

Module scripts[p1153] can be classified into five types:

- A module script[p1153] is a **JavaScript module script** if its record[p1152] is a Source Text Module Record.
- A module script[p1153] is a **CSS module script** if its record[p1152] is a Synthetic Module Record, and it was created via the create a CSS module script[p1163] algorithm. CSS module scripts represent a parsed CSS style sheet.
- A module script[p1153] is a **JSON module script** if its record[p1152] is a Synthetic Module Record, and it was created via the create a JSON module script[p1164] algorithm. JSON module scripts represent a parsed JSON document.
- A module script[p1153] is a **text module script** if its record[p1152] is a Synthetic Module Record, and it was created via the create a text module script[p1164] algorithm. Text module scripts represent textual data encoded as UTF-8.
- A module script[p1153] is a **WebAssembly module script** if its record[p1152] is a WebAssembly Module Record.

Note

As CSS style sheets, JSON documents, and text do not import dependent modules, and do not throw exceptions on evaluation, the fetch options[p1153] and base URL[p1153] of CSS module scripts[p1153], JSON module scripts[p1153], and text module scripts[p1153] are always null.

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]].

Note

The active script[p1154] concept is so far only used by the `import()` feature, to determine the base URL[p1153] to use for resolving relative module specifiers.

8.1.4.2 Fetching scripts §[p1154]

This section introduces a number of algorithms for fetching scripts, taking various necessary inputs and resulting in classic[p1153] or module scripts[p1153].

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[p1153]) and for fetching any imported modules (for both module scripts[p1153] and classic scripts[p1153])

referrer policy
: The referrer policy used for the initial fetch and for fetching any imported modules

Note

This policy can mutate after a module script[p1153]'s response is received, to be the referrer policy parsed from the response, and used when fetching any module dependencies.

render-blocking
: The boolean value of render-blocking used for the initial fetch and for fetching any imported modules. Unless otherwise stated, its value is false.

fetch priority
: The priority used for the initial fetch

Note

Recall that via the `import()` feature, classic scripts[p1153] can import module scripts[p1153].

The **default script fetch options** are a script fetch options[p1154] whose cryptographic nonce[p1154] is the empty string, integrity metadata[p1154] is the empty string, parser metadata[p1154] is "`not-parser-inserted`", credentials mode[p1154] is "`same-origin`", referrer policy[p1154] is the empty string, and fetch priority[p1154] is "`auto`".

To **set up the classic script request**, given a request *request* and a script fetch options[p1154] *options*, set *request*'s cryptographic nonce metadata to *options*'s cryptographic nonce[p1154], its integrity metadata to *options*'s integrity metadata[p1154], its parser metadata to *options*'s parser metadata[p1154], its referrer policy to *options*'s referrer policy[p1154], its render-blocking to *options*'s render-blocking[p1154], and its priority to *options*'s fetch priority[p1154].

To **set up the module script request**, given a request *request* and a script fetch options[p1154] *options*, set *request*'s cryptographic nonce metadata to *options*'s cryptographic nonce[p1154], its integrity metadata to *options*'s integrity metadata[p1154], its parser metadata to *options*'s parser metadata[p1154], its credentials mode to *options*'s credentials mode[p1154], its referrer policy to *options*'s referrer policy[p1154], its render-blocking to *options*'s render-blocking[p1154], and its priority to *options*'s fetch priority[p1154].

To **get the descendant script fetch options** given a script fetch options[p1154] *originalOptions*, a URL *url*, and an environment settings object[p1144] *settingsObject*:

1. Let *newOptions* be a copy of *originalOptions*.
2. Let *integrity* be the result of resolving a module integrity metadata[p1155] with *url* and *settingsObject*.
3. Set *newOptions*'s integrity metadata[p1154] to *integrity*.
4. Set *newOptions*'s fetch priority[p1154] to "`auto`".
5. Return *newOptions*.

To **resolve a module integrity metadata**, given a URL *url* and an environment settings object[p1144] *settingsObject*:

1. Let *map* be *settingsObject*'s global object[p1145]'s import map[p1145].
2. If *map*'s integrity[p1177][*url*] does not exist, then return the empty string.
3. Return *map*'s integrity[p1177][*url*].

Several of the below algorithms can be customized with a **perform the fetch hook** algorithm, which takes a request, a boolean *isTopLevel*[p1155], and a ***processCustomFetchResponse*** algorithm. It runs *processCustomFetchResponse*[p1155] with a response and either null (on failure) or a byte sequence containing the response body. ***isTopLevel*** will be true for all classic script[p1153] fetches, and for the initial fetch when fetching an external module script graph[p1157] or fetching a module worker script graph[p1158], but false for the fetches resulting from `import` statements encountered throughout the graph or from `import()` expressions.

By default, not supplying a perform the fetch hook[p1155] will cause the below algorithms to simply fetch the given request, with algorithm-specific customizations to the request and validations of the resulting response.

To layer your own customizations on top of these algorithm-specific ones, supply a perform the fetch hook[p1155] that modifies the given request, fetches it, and then performs specific validations of the resulting response (completing with a network error if the validations fail).

The hook can also be used to perform more subtle customizations, such as keeping a cache of responses and avoiding performing a fetch at all.

Note

Service Workers *is an example of a specification that runs these algorithms with its own options for the hook. [SW][p1587]*

Now for the algorithms themselves.

To **fetch a classic script** given a URL *url*, an environment settings object[p1144] *settingsObject*, a script fetch options[p1154] *options*, a CORS settings attribute state[p103] *corsSetting*, an encoding *encoding*, and an algorithm *onComplete*, run these steps. *onComplete* must be an algorithm accepting null (on failure) or a classic script[p1153] (on success).

1. Let *request* be the result of creating a potential-CORS request[p102] given *url*, "`script`", and *corsSetting*.
2. Set *request*'s client to *settingsObject*.
3. Set *request*'s initiator type to "`script`".
4. Set up the classic script request[p1154] given *request* and *options*.
5. Fetch *request* with the following *processResponseConsumeBody* steps given response *response* and null, failure, or a byte sequence *bodyBytes*:

Note

response *can be either CORS-same-origin[p102] or CORS-cross-origin[p102]. This only affects how error reporting happens.*

1. Set *response* to *response*'s unsafe response[p102].
2. If any of the following are true:
    - *bodyBytes* is null or failure; or
    - *response*'s status is not an ok status,

    then run *onComplete* given null, and abort these steps.

    Note

    For historical reasons, this algorithm does not include MIME type checking, unlike the other script-fetching algorithms in this section.
3. Let *potentialMIMETypeForEncoding* be the result of extracting a MIME type given *response*'s header list.
4. Set *encoding* to the result of legacy extracting an encoding given *potentialMIMETypeForEncoding* and *encoding*.

    Note

    This intentionally ignores the MIME type essence.
5. Let *sourceText* be the result of decoding *bodyBytes* to Unicode, using *encoding* as the fallback encoding.

    Note

    The decode algorithm overrides encoding *if the file contains a BOM.*
6. Let *mutedErrors* be true if *response* was CORS-cross-origin[p102], and false otherwise.
7. Let *script* be the result of creating a classic script[p1161] given *sourceText*, *settingsObject*, *response*'s URL, *options*, *mutedErrors*, and *url*.
8. Run *onComplete* given *script*.

To **fetch a classic worker script** given a URL *url*, an environment settings object[p1144] *fetchClient*, a destination *destination*, an environment settings object[p1144] *settingsObject*, an algorithm *onComplete*, and an optional perform the fetch hook[p1155] *performFetch*, run these steps. *onComplete* must be an algorithm accepting null (on failure) or a classic script[p1153] (on success).

1. Let *request* be a new request whose URL is *url*, client is *fetchClient*, destination is *destination*, initiator type is "`other`", 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 *performFetch* was given, run *performFetch* with *request*, true, and with *processResponseConsumeBody* as defined below.

    Otherwise, fetch *request* with *processResponseConsumeBody* set to *processResponseConsumeBody* as defined below.

    In both cases, let *processResponseConsumeBody* given response *response* and null, failure, or a byte sequence *bodyBytes* be the following algorithm:

    1. Set *response* to *response*'s unsafe response[p102].
    2. If any of the following are true:
        - *bodyBytes* is null or failure; or
        - *response*'s status is not an ok status,

        then run *onComplete* given null, and abort these steps.
    3. If all of the following are true:
        - *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 run *onComplete* given null, and abort these steps.

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.

4. Let *sourceText* be the result of UTF-8 decoding *bodyBytes*.
5. Let *script* be the result of creating a classic script[p1161] using *sourceText*, *settingsObject*, *response*'s URL, and the default script fetch options[p1154].
6. Run *onComplete* given *script*.

To **fetch a classic worker-imported script** given a URL *url*, an environment settings object[p1144] *settingsObject*, and an optional perform the fetch hook[p1155] *performFetch*, run these steps. The algorithm will return a classic script[p1153] on success, or throw an exception on failure.

1. Let *response* be null.
2. Let *bodyBytes* be null.
3. Let *request* be a new request whose URL is *url*, client is *settingsObject*, destination is "`script`", initiator type is "`other`", parser metadata is "`not parser-inserted`", and whose use-URL-credentials flag is set.
4. If *performFetch* was given, run *performFetch* with *request*, *isTopLevel*, and with *processResponseConsumeBody* as defined below.

   Otherwise, fetch *request* with *processResponseConsumeBody* set to *processResponseConsumeBody* as defined below.

   In both cases, let *processResponseConsumeBody* given response *res* and null, failure, or a byte sequence *bb* be the following algorithm:

   1. Set *bodyBytes* to *bb*.
   2. Set *response* to *res*.
5. Pause[p1203] until *response* is not null.

   Note

   Unlike other algorithms in this section, the fetching process is synchronous here.
6. Set *response* to *response*'s unsafe response[p102].
7. If any of the following are true:

   - *bodyBytes* is null or failure;
   - *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`.
8. Let *sourceText* be the result of UTF-8 decoding *bodyBytes*.
9. Let *mutedErrors* be true if *response* was CORS-cross-origin[p102], and false otherwise.
10. Let *script* be the result of creating a classic script[p1161] given *sourceText*, *settingsObject*, *response*'s URL, the default script fetch options[p1154], and *mutedErrors*.
11. Return *script*.

To **fetch an external module script graph** given a URL *url*, an environment settings object[p1144] *settingsObject*, a script fetch options[p1154] *options*, and an algorithm *onComplete*, run these steps. *onComplete* must be an algorithm accepting null (on failure) or a module script[p1153] (on success).

1. Fetch a single module script[p1160] given *url*, *settingsObject*, "`script`", *options*, *settingsObject*, "`client`", true, and with the following steps given *result*:

   1. If *result* is null, run *onComplete* given null, and abort these steps.

2. Fetch the descendants of and link[p1159] *result* given *settingsObject*, "`script`", and *onComplete*.

To **fetch a modulepreload module script graph** given a URL *url*, a destination *destination*, an environment settings object[p1144] *settingsObject*, a script fetch options[p1154] *options*, and an algorithm *onComplete*, run these steps. *onComplete* must be an algorithm accepting null (on failure) or a module script[p1153] (on success).

1. Fetch a single module script[p1160] given *url*, *settingsObject*, *destination*, *options*, *settingsObject*, "`client`", true, and with the following steps given *result*:

    1. Run *onComplete* given *result*.

    2. Assert: *settingsObject*'s global object[p1145] implements `Window`[p965].

    3. If *result* is not null, optionally fetch the descendants of and link[p1159] *result* given *settingsObject*, *destination*, and an empty algorithm.

    Note

    Generally, performing this step 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[p1157] 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 string *sourceText*, a URL *baseURL*, an environment settings object[p1144] *settingsObject*, a script fetch options[p1154] *options*, and an algorithm *onComplete*, run these steps. *onComplete* must be an algorithm accepting null (on failure) or a module script[p1153] (on success).

1. Let *script* be the result of creating a JavaScript module script[p1162] using *sourceText*, *settingsObject*, *baseURL*, and *options*.

2. Fetch the descendants of and link[p1159] *script*, given *settingsObject*, "`script`", and *onComplete*.

To **fetch a module worker script graph** given a URL *url*, an environment settings object[p1144] *fetchClient*, a destination *destination*, a credentials mode[p1154] *credentialsMode*, an environment settings object[p1144] *settingsObject*, and an algorithm *onComplete*, fetch a worklet/module worker script graph[p1159] given *url*, *fetchClient*, *destination*, *credentialsMode*, *settingsObject*, and *onComplete*.

To **fetch a worklet script graph** given a URL *url*, an environment settings object[p1144] *fetchClient*, a destination *destination*, a credentials mode[p1154] *credentialsMode*, an environment settings object[p1144] *settingsObject*, a module responses map[p1345] *moduleResponsesMap*, and an algorithm *onComplete*, fetch a worklet/module worker script graph[p1159] given *url*, *fetchClient*, *destination*, *credentialsMode*, *settingsObject*, *onComplete*, and the following perform the fetch hook[p1155] given *request* and *processCustomFetchResponse*[p1155]:

1. Let *requestURL* be *request*'s URL.

2. If *moduleResponsesMap*[*requestURL*] is "`fetching`", wait in parallel[p44] until that entry's value changes, then queue a task[p1194] on the networking task source[p1204] to proceed with running the following steps.

3. If *moduleResponsesMap*[*requestURL*] exists:

    1. Let *cached* be *moduleResponsesMap*[*requestURL*].

    2. Run *processCustomFetchResponse* with *cached*[0] and *cached*[1].

    3. Return.

4. Set *moduleResponsesMap*[*requestURL*] to "`fetching`".

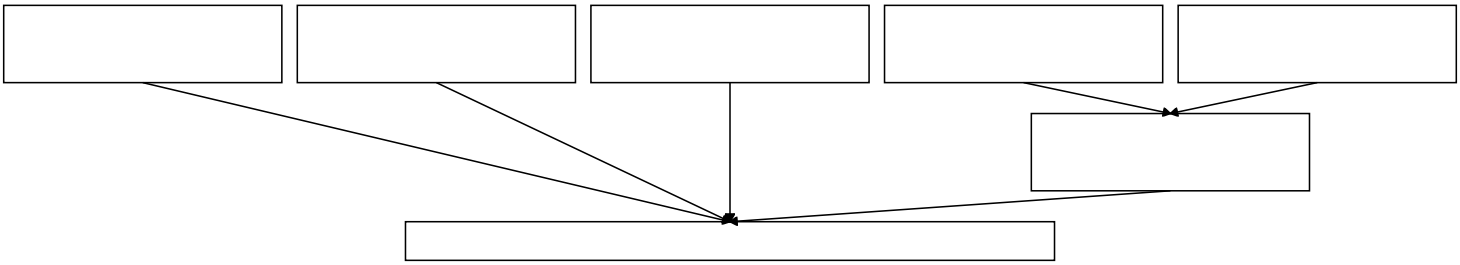
5. Fetch *request*, with *processResponseConsumeBody* set to the following steps given response *response* and null, failure, or a byte sequence *bodyBytes*:

    1. Set *moduleResponsesMap*[*requestURL*] to (*response*, *bodyBytes*).

    2. Run *processCustomFetchResponse* with *response* and *bodyBytes*.

The following algorithms are meant for internal use by this specification only as part of fetching an external module script graph[p1157] 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 *url*, an environment settings object[p1144] *fetchClient*, a destination *destination*, a credentials mode[p1154] *credentialsMode*, an environment settings object[p1144] *settingsObject*, an algorithm *onComplete*, and an optional perform the fetch hook[p1155] *performFetch*, run these steps. *onComplete* must be an algorithm accepting null (on failure) or a module script[p1153] (on success).

1. Let *options* be a script fetch options[p1154] whose cryptographic nonce[p1154] is the empty string, integrity metadata[p1154] is the empty string, parser metadata[p1154] is "`not-parser-inserted`", credentials mode[p1154] is *credentialsMode*, referrer policy[p1154] is the empty string, and fetch priority[p1154] is "`auto`".

2. Fetch a single module script[p1160] given *url*, *fetchClient*, *destination*, *options*, *settingsObject*, "`client`", true, and *onSingleFetchComplete* as defined below. If *performFetch* was given, pass it along as well.

   onSingleFetchComplete given *result* is the following algorithm:

   1. If *result* is null, run *onComplete* given null, and abort these steps.

   2. Fetch the descendants of and link[p1159] *result* given *fetchClient*, *destination*, and *onComplete*. If *performFetch* was given, pass it along as well.

To **fetch the descendants of and link** a module script[p1153] *moduleScript*, given an environment settings object[p1144] *fetchClient*, a destination *destination*, an algorithm *onComplete*, and an optional perform the fetch hook[p1155] *performFetch*, run these steps. *onComplete* must be an algorithm accepting null (on failure) or a module script[p1153] (on success).

1. Let *record* be *moduleScript*'s record[p1152].

2. If *record* is null:

   1. Set *moduleScript*'s error to rethrow[p1153] to *moduleScript*'s parse error[p1370].

   2. Run *onComplete* given *moduleScript*.

   3. Return.

3. Let *state* be Record { [[ErrorToRethrow]]: null, [[Destination]]: *destination*, [[PerformFetch]]: null, [[FetchClient]]: *fetchClient* }.

4. If *performFetch* was given, set *state*.[[PerformFetch]] to *performFetch*.

5. Let *loadingPromise* be *record*.LoadRequestedModules(*state*).

   Note

   This step will recursively load all the module transitive dependencies.

6. Upon fulfillment of *loadingPromise*, run the following steps:

   1. Perform *record*.Link().

      Note

      This step will recursively call Link on all of the module's unlinked dependencies.

      If this throws an exception, catch it, and set *moduleScript*'s error to rethrow[p1153] to that exception.

   2. Run *onComplete* given *moduleScript*.

7. Upon rejection of *loadingPromise*, run the following steps:

   1. If *state*.[[ErrorToRethrow]] is not null, set *moduleScript*'s error to rethrow[p1153] to *state*.[[ErrorToRethrow]] and run *onComplete* given *moduleScript*.

2. Otherwise, run *onComplete* given null.

Note

state.*[[ErrorToRethrow]] is null when* loadingPromise *is rejected due to a loading error.*

To **fetch a single module script**, given a URL *url*, an environment settings object[p1144] *fetchClient*, a destination *destination*, a script fetch options[p1154] *options*, an environment settings object[p1144] *settingsObject*, a referrer *referrer*, an optional ModuleRequest Record *moduleRequest*, a boolean *isTopLevel*[p1155], an algorithm *onComplete*, and an optional perform the fetch hook[p1155] *performFetch*, run these steps. *onComplete* must be an algorithm accepting null (on failure) or a module script[p1153] (on success).

1. Let *moduleType* be "`javascript-or-wasm`".
2. If *moduleRequest* was given, then set *moduleType* to the result of running the module type from module request[p1164] steps given *moduleRequest*.
3. Assert: the result of running the module type allowed[p1164] steps given *moduleType* and *settingsObject* is true. Otherwise, we would not have reached this point because a failure would have been raised when inspecting *moduleRequest*.[[Attributes]] in HostLoadImportedModule[p1191] or fetch a single imported module script[p1161].
4. Let *moduleMap* be *settingsObject*'s module map[p1144].
5. If *moduleMap*[(*url*, *moduleType*)] is a module script[p1153], run *onComplete* given *moduleMap*[(*url*, *moduleType*)], and return.
6. If *moduleMap*[(*url*, *moduleType*)] is a list, append *onComplete* to *moduleMap*[(*url*, *moduleType*)], and return.
7. Set *moduleMap*[(*url*, *moduleType*)] to « *onComplete* ».
8. Let *request* be a new request whose URL is *url*, mode is "`cors`", referrer is *referrer*, and client is *fetchClient*.
9. Set *request*'s destination to the result of running the fetch destination from module type[p1164] steps given *destination* and *moduleType*.
10. If *destination* is "`worker`", "`sharedworker`", or "`serviceworker`", and *isTopLevel* is true, then set *request*'s mode to "`same-origin`".
11. Set *request*'s initiator type to "`script`".
12. Set up the module script request[p1155] given *request* and *options*.
13. If *performFetch* was given, run *performFetch* with *request*, *isTopLevel*, and with *processResponseConsumeBody* as defined below.

    Otherwise, fetch *request* with *processResponseConsumeBody* set to *processResponseConsumeBody* as defined below.

    In both cases, let *processResponseConsumeBody* given response *response* and null, failure, or a byte sequence *bodyBytes* be the following algorithm:

    Note

    response *is always CORS-same-origin*[p102].

    1. If any of the following are true:
       - *bodyBytes* is null or failure; or
       - *response*'s status is not an ok status,

       then:
       1. Let *callbacks* be *moduleMap*[(*url*, *moduleType*)].
       2. Remove *moduleMap*[(*url*, *moduleType*)].
       3. For each *callback* of *callbacks*: run *callback* given null.
       4. Return.
    2. Let *mimeType* be the result of extracting a MIME type from *response*'s header list.
    3. Let *moduleScript* be null.

4. Let *referrerPolicy* be the result of parsing the `Referrer-Policy` header given *response*. [REFERRERPOLICY]

5. If *referrerPolicy* is not the empty string, set *options*'s referrer policy to *referrerPolicy*.

6. If *mimeType*'s essence is "application/wasm" and *moduleType* is "javascript-or-wasm", then set *moduleScript* to the result of creating a WebAssembly module script given *bodyBytes*, *settingsObject*, *response*'s URL, and *options*.

7. Otherwise:

    1. Let *sourceText* be the result of UTF-8 decoding *bodyBytes*.

    2. If *moduleType* is "text", then set *moduleScript* to the result of creating a text module script given *sourceText* and *settingsObject*.

    3. If *mimeType* is a JavaScript MIME type and *moduleType* is "javascript-or-wasm", then set *moduleScript* to the result of creating a JavaScript module script given *sourceText*, *settingsObject*, *response*'s URL, and *options*.

    4. If the MIME type essence of *mimeType* is "text/css" and *moduleType* is "css", then set *moduleScript* to the result of creating a CSS module script given *sourceText* and *settingsObject*.

    5. If *mimeType* is a JSON MIME type and *moduleType* is "json", then set *moduleScript* to the result of creating a JSON module script given *sourceText* and *settingsObject*.

8. Let *callbacks* be *moduleMap*[(*url*, *moduleType*)].

9. If *moduleScript* is null, then remove *moduleMap*[(*url*, *moduleType*)]; otherwise set *moduleMap*[(*url*, *moduleType*)] to *moduleScript*.

    Note

    It is intentional that the module map is keyed by the request URL, whereas the base URL for the module script is set to the response URL. The former is used to deduplicate fetches, while the latter is used for URL resolution.

10. For each *callback* of *callbacks*: run *callback* given *moduleScript*.

To **fetch a single imported module script**, given a URL *url*, an environment settings object *fetchClient*, a destination *destination*, a script fetch options *options*, environment settings object *settingsObject*, a referrer *referrer*, a ModuleRequest Record *moduleRequest*, an algorithm *onComplete*, and an optional perform the fetch hook *performFetch*, run these steps. *onComplete* must be an algorithm accepting null (on failure) or a module script (on success).

1. Assert: *moduleRequest*.[[Attributes]] does not contain any Record *entry* such that *entry*.[[Key]] is not "type", because we only asked for "type" attributes in HostGetSupportedImportAttributes.

2. Let *moduleType* be the result of running the module type from module request steps given *moduleRequest*.

3. If the result of running the module type allowed steps given *moduleType* and *settingsObject* is false, then run *onComplete* given null, and return.

4. Fetch a single module script given *url*, *fetchClient*, *destination*, *options*, *settingsObject*, *referrer*, *moduleRequest*, false, and *onComplete*. If *performFetch* was given, pass it along as well.

8.1.4.3 Creating scripts §p11 61

This standard refers to the following concepts defined there:

- **WebDriver BiDi command "script.evaluate"**

To **create a classic script**, given a string *source*, an environment settings object *settings*, a URL *baseURL*, a script fetch options *options*, an optional boolean *mutedErrors* (default false), an optional URL-or-null *sourceURLForWindowScripts* (default null), and an optional boolean *bypassDisabledScripting* (default false):

Note

The bypassDisabledScripting *parameter is intended to be used for running scripts even if scripting is disabled. This is required*

for some automation scenarios, e.g. for WebDriver BiDi command "script.evaluate".

Note

The bypassDisabledScripting *parameter is intended to be used for running scripts even if scripting is disabled. This is required for some automation scenarios, e.g. for WebDriver BiDi command "script.evaluate". When the scripting process (e.g., executing a script via a WebDriver BiDi command) is invoked with* bypassDisabledScripting *set to true, the event loop is allowed to run to resolve thenables (from promises) and weakmap finalizations created by that script, even if scripting is disabled[p1152]. This is necessary to ensure the proper functioning of testing scripts, which often rely on asynchronous operations.*

1. If *mutedErrors* is true, then set *baseURL* to `about:blank`[p54].

   Note

   When mutedErrors *is true,* baseURL *is the script's CORS-cross-origin[p102] response's URL, which shouldn't be exposed to JavaScript. Therefore,* baseURL *is sanitized here.*

2. If scripting is disabled[p1152] for *settings* and *bypassDisabledScripting* is false, then set *source* to the empty string.
3. Let *script* be a new classic script[p1153] that this algorithm will subsequently initialize.
4. Set *script*'s settings object[p1152] to *settings*.
5. Set *script*'s base URL[p1153] to *baseURL*.
6. Set *script*'s fetch options[p1153] to *options*.
7. Set *script*'s muted errors[p1153] to *mutedErrors*.
8. Set *script*'s parse error[p1153] and error to rethrow[p1153] to null.
9. Record classic script creation time given *script* and *sourceURLForWindowScripts*.
10. Let *result* be ParseScript(*source*, *settings*'s realm[p1145], *script*).

    Note

    Passing script *as the last parameter here ensures* result.*[[HostDefined]] will be* script.

11. If *result* is a list of errors:

    1. Set *script*'s parse error[p1153] and its error to rethrow[p1153] to *result*[0].
    2. Return *script*.

12. Set *script*'s record[p1152] to *result*.
13. Return *script*.

To **create a JavaScript module script**, given a string *source*, an environment settings object[p1144] *settings*, a URL *baseURL*, and a script fetch options[p1154] *options*:

1. If scripting is disabled[p1152] for *settings*, then set *source* to the empty string.
2. Let *script* be a new module script[p1153] that this algorithm will subsequently initialize.
3. Set *script*'s settings object[p1152] to *settings*.
4. Set *script*'s base URL[p1153] to *baseURL*.
5. Set *script*'s fetch options[p1153] to *options*.
6. Set *script*'s parse error[p1153] and error to rethrow[p1153] to null.
7. Let *result* be ParseModule(*source*, *settings*'s realm[p1145], *script*).

   Note

   Passing script *as the last parameter here ensures* result.*[[HostDefined]] will be* script.

8. If *result* is a list of errors:

    1. Set *script*'s parse error[p1153] to *result*[0].

    2. Return *script*.

9. Set *script*'s record[p1152] to *result*.

10. Return *script*.

To **create a WebAssembly module script**, given a byte sequence *bodyBytes*, an environment settings object[p1144] *settings*, a URL *baseURL*, and a script fetch options[p1154] *options*:

1. If scripting is disabled[p1152] for *settings*, then set *bodyBytes* to the byte sequence 0x00 0x61 0x73 0x6D 0x01 0x00 0x00 0x00.

    Note

    This byte sequence corresponds to an empty WebAssembly module with only the magic bytes and version number provided.

2. Let *script* be a new module script[p1153] that this algorithm will subsequently initialize.

3. Set *script*'s settings object[p1152] to *settings*.

4. Set *script*'s base URL[p1153] to *baseURL*.

5. Set *script*'s fetch options[p1153] to *options*.

6. Set *script*'s parse error[p1153] and error to rethrow[p1153] to null.

7. Let *result* be the result of parsing a WebAssembly module given *bodyBytes*, *settings*'s realm[p1145], and *script*.

    Note

    Passing script *as the last parameter here ensures* result.*[[HostDefined]] will be* script.

8. If the previous step threw an error *error*:

    1. Set *script*'s parse error[p1153] to *error*.

    2. Return *script*.

9. Set *script*'s record[p1152] to *result*.

10. Return *script*.

Note

WebAssembly JavaScript Interface: ESM Integration *specifies the hooks for the WebAssembly integration with ECMA-262 module loading. This includes support both for direct dependency imports, as well as for source phase imports, which support virtualization and multi-instantiation. [WASMESM][p1587]*

To **create a CSS module script**, given a string *source* and an environment settings object[p1144] *settings*:

1. Let *script* be a new module script[p1153] that this algorithm will subsequently initialize.

2. Set *script*'s settings object[p1152] to *settings*.

3. Set *script*'s base URL[p1153] and fetch options[p1153] to null.

4. Set *script*'s parse error[p1153] and error to rethrow[p1153] to null.

5. Let *sheet* be the result of running the steps to create a constructed `CSSStyleSheet` with an empty dictionary as the argument.

6. Run the steps to synchronously replace the rules of a `CSSStyleSheet` on *sheet* given *source*.

    If this throws an exception, catch it, and set *script*'s parse error[p1153] to that exception, and return *script*.

Note

The steps to synchronously replace the rules of a `CSSStyleSheet` *will throw if* source *contains any* `@import` *rules. This is by-design for now because there is not yet an agreement on how to handle these for CSS module scripts; therefore they are blocked altogether until a consensus is reached.*

7. Set *script*'s record[p1152] to the result of CreateDefaultExportSyntheticModule(*sheet*).
8. Return *script*.

To **create a JSON module script**, given a string *source* and an environment settings object[p1144] *settings*:

1. Let *script* be a new module script[p1153] that this algorithm will subsequently initialize.
2. Set *script*'s settings object[p1152] to *settings*.
3. Set *script*'s base URL[p1153] and fetch options[p1153] to null.
4. Set *script*'s parse error[p1153] and error to rethrow[p1153] to null.
5. Let *result* be ParseJSONModule(*source*).

   If this throws an exception, catch it, and set *script*'s parse error[p1153] to that exception, and return *script*.
6. Set *script*'s record[p1152] to *result*.
7. Return *script*.

To **create a text module script**, given a string *text* and an environment settings object[p1144] *settings*:

1. Let *script* be a new module script[p1153] that this algorithm will subsequently initialize.
2. Set *script*'s settings object[p1152] to *settings*.
3. Set *script*'s base URL[p1153] and fetch options[p1153] to null.
4. Set *script*'s parse error[p1153] and error to rethrow[p1153] to null.
5. Let *result* be CreateTextModule(*text*).
6. Set *script*'s record[p1152] to *result*.
7. Return *script*.

The **module type from module request** steps, given a ModuleRequest Record *moduleRequest*, are as follows:

1. Let *moduleType* be "`javascript-or-wasm`".
2. If *moduleRequest*.[[Attributes]] has a Record *entry* such that *entry*.[[Key]] is "`type`":
   1. If *entry*.[[Value]] is "`javascript-or-wasm`", then set *moduleType* to null.

      Note

      This specification uses the "`javascript-or-wasm`" *module type internally for JavaScript module scripts[p1153] or WebAssembly module scripts[p1153], so this step is needed to prevent modules from being imported using a* "`javascript-or-wasm`" *type attribute (a null* moduleType *will cause the module type allowed[p1164] check to fail).*
   2. Otherwise, set *moduleType* to *entry*.[[Value]].
3. Return *moduleType*.

The **module type allowed** steps, given a string *moduleType* and an environment settings object[p1144] *settings*, are as follows:

1. If *moduleType* is not "`javascript-or-wasm`", "`css`", "`json`", or "`text`", then return false.
2. If *moduleType* is "`css`" and the `CSSStyleSheet` interface is not exposed in *settings*'s realm[p1145], then return false.
3. Return true.

The **fetch destination from module type** steps, given a destination *defaultDestination* and a string *moduleType*, are as follows:

1. If *moduleType* is "`json`", then return "`json`".
2. If *moduleType* is "`css`", then return "`style`".
3. If *moduleType* is "`text`", then return "`text`".
4. Return *defaultDestination*.

8.1.4.4 Calling scripts §[p1165]

To **run a classic script** given a classic script[p1153] *script* and an optional boolean *rethrow errors* (default false):

1. Let *settings* be the settings object[p1152] of *script*.
2. Check if we can run script[p1166] with *settings*. If this returns "do not run", then return NormalCompletion(empty).
3. Record classic script execution start time given *script*.
4. Prepare to run script[p1166] given *settings*.
5. Let *evaluationStatus* be null.
6. If *script*'s error to rethrow[p1153] is not null, then set *evaluationStatus* to ThrowCompletion(*script*'s error to rethrow[p1153]).
7. Otherwise, set *evaluationStatus* to ScriptEvaluation(*script*'s record[p1152]).

   If ScriptEvaluation does not complete because the user agent has aborted the running script[p1166], leave *evaluationStatus* as null.
8. If *evaluationStatus* is an abrupt completion:
    1. If *rethrow errors* is true and *script*'s muted errors[p1153] is false:
        1. Clean up after running script[p1166] with *settings*.
        2. Rethrow *evaluationStatus*.[[Value]].
    2. If *rethrow errors* is true and *script*'s muted errors[p1153] is true:
        1. Clean up after running script[p1166] with *settings*.
        2. Throw a "`NetworkError`" `DOMException`.
    3. Otherwise, *rethrow errors* is false. Perform the following steps:
        1. Report an exception[p1167] given by *evaluationStatus*.[[Value]] for *script*'s settings object[p1152]'s global object[p1145].
        2. Clean up after running script[p1166] with *settings*.
        3. Return *evaluationStatus*.
9. Clean up after running script[p1166] with *settings*.
10. If *evaluationStatus* is a normal completion, then return *evaluationStatus*.
11. If we've reached this point, *evaluationStatus* was left as null because the script was aborted prematurely[p1166] during evaluation. Return ThrowCompletion(a new `QuotaExceededError`).

To **run a module script** given a module script[p1153] *script* and an optional boolean *preventErrorReporting* (default false):

1. Let *settings* be the settings object[p1152] of *script*.
2. Check if we can run script[p1166] with *settings*. If this returns "do not run", then return a promise resolved with undefined.
3. Record module script execution start time given *script*.
4. Prepare to run script[p1166] given *settings*.
5. Let *evaluationPromise* be null.

6. If *script*'s error to rethrow[p1153] is not null, then set *evaluationPromise* to a promise rejected with *script*'s error to rethrow[p1153].

7. Otherwise:

    1. Let *record* be *script*'s record[p1152].

    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[p1166], then set *evaluationPromise* to a promise rejected with a new `QuotaExceededError`.

8. If *preventErrorReporting* is false, then upon rejection of *evaluationPromise* with *reason*, report an exception[p1167] given by *reason* for *script*'s settings object[p1152]'s global object[p1145].

9. Clean up after running script[p1166] with *settings*.

10. Return *evaluationPromise*.

The steps to **check if we can run script** with an environment settings object[p1144] *settings* are as follows. They return either "run" or "do not run".

1. If the global object[p1145] specified by *settings* is a `Window`[p965] object whose `Document`[p135] object is not fully active[p1051], then return "do not run".

2. If scripting is disabled[p1152] for *settings*, then return "do not run".

3. Return "run".

The steps to **prepare to run script** with an environment settings object[p1144] *settings* are as follows:

1. Push *settings*'s realm execution context[p1144] onto the JavaScript execution context stack; it is now the running JavaScript execution context.

2. Add *settings* to the surrounding agent's event loop[p1193]'s currently running task[p1194]'s script evaluation environment settings object set[p1194].

The steps to **clean up after running script** with an environment settings object[p1144] *settings* are as follows:

1. Assert: *settings*'s realm execution context[p1144] is the running JavaScript execution context.

2. Remove *settings*'s realm execution context[p1144] from the JavaScript execution context stack.

3. If the JavaScript execution context stack is now empty, perform a microtask checkpoint[p1200]. (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[p1152] in the [[HostDefined]] field in the ScriptOrModule component of the running JavaScript execution context.

8.1.4.5 Killing scripts §[p1166]

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][p1583]

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`, abort the script[p1166] 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.

```
<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()` [p1259] 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 [p1046]. This is a distinct operation from closing the page, since it might lead to the user being presented with new content rather than the page being closed.

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` [p1576] event handlers.

✔ MDN

8.1.4.6 Runtime script errors §[p1167]

For web developers (non-normative)

`self.reportError` [p1168]**`(e)`**

Dispatches an `error` [p1575] event at the global object for the given value *e*, in the same fashion as an unhandled exception.

To **extract error information** from a JavaScript value *exception*:

1. Let *attributes* be an empty map keyed by IDL attributes.
2. Set *attributes*[`error` [p1169]] to *exception*.
3. Set *attributes*[`message` [p1169]], *attributes*[`filename` [p1169]], *attributes*[`lineno` [p1169]], and *attributes*[`colno` [p1169]] to implementation-defined values derived from *exception*.

Note

Browsers implement behavior not specified here or in the JavaScript specification to gather values which are helpful, including in unusual cases (e.g., `eval`). In the future, this might be specified in greater detail.

4. Return *attributes*.

To **report an exception** *exception* which is a JavaScript value, for a particular global object [p1145] *global* and optional boolean ***omitError*** (default false):

1. Let *notHandled* be true.
2. Let *errorInfo* be the result of extracting error information [p1167] from *exception*.
3. Let *script* be a script [p1152] found in an implementation-defined way, or null. This should usually be the running script [p1166] (most notably during run a classic script [p1165]).

Note

Implementations have not yet settled on interoperable behavior for which script is used to determine whether errors are muted in less common cases.

4. If *script* is a classic script [p1153] and *script*'s muted errors [p1153] is true, then set *errorInfo*[`error` [p1169]] to null, *errorInfo*[`message` [p1169]] to "`Script error.`", *errorInfo*[`filename` [p1169]] to the empty string, *errorInfo*[`lineno` [p1169]] to 0, and *errorInfo*[`colno` [p1169]] to 0.
5. If *omitError* is true, then set *errorInfo*[`error` [p1169]] to null.

6. If *global* is not in error reporting mode[p1145]:

    1. Set *global*'s in error reporting mode[p1145] to true.

    2. If *global* implements EventTarget, then set *notHandled* to the result of firing an event named error[p1575] at *global*, using ErrorEvent[p1168], with the cancelable attribute initialized to true, and additional attributes initialized according to *errorInfo*.

        Note

        Returning true in an event handler cancels the event per the event handler processing algorithm[p1210].

    3. Set *global*'s in error reporting mode[p1145] to false.

7. If *notHandled* is true:

    1. Set *errorInfo*[error[p1169]] to null.

    2. If *global* implements DedicatedWorkerGlobalScope[p1324], queue a global task[p1195] on the DOM manipulation task source[p1203] with the *global*'s associated Worker[p1332]'s relevant global object[p1151] to run these steps:

        1. Let *workerObject* be the Worker[p1332] object associated with *global*.

        2. Set *notHandled* to the result of firing an event named error[p1575] at *workerObject*, using ErrorEvent[p1168], with the cancelable attribute initialized to true, and additional attributes initialized according to *errorInfo*.

        3. If *notHandled* is true, then report[p1167] *exception* for *workerObject*'s relevant global object[p1151] with *omitError*[p1167] set to true.

            Note

            The actual exception *value will not be available in the owner realm, but the user agent still carries through enough information to set the message, filename, and other attributes, as well as potentially report to a developer console.*

    3. Otherwise, the user agent may report *exception* to a developer console.

If the implicit port connecting a worker to its Worker[p1332] object has been disentangled (i.e. if the parent worker has been terminated), then the user agent must act as if the Worker[p1332] object had no error[p1575] event handler and as if that worker's onerror[p1323] 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[p135], even if some of the workers along this chain have been terminated and garbage collected.

Previous revisions of this standard defined an algorithm to **report the exception**. As part of issue #958, this has been superseded by report an exception[p1167] which behaves differently and takes different inputs. Issue #10516 tracks updating the specification ecosystem.

The **reportError(e)** method steps are to report an exception[p1167] *e* for this.

It is unclear whether muting[p1153] is applicable here. In Chrome and Safari it is muted, but in Firefox it is not. See also issue #958.

The ErrorEvent[p1168] interface is defined as follows:

IDL

```
[Exposed=*]
interface ErrorEvent : Event {
  constructor(DOMString type, optional ErrorEventInit eventInitDict = {});

  readonly attribute DOMString message;
  readonly attribute USVString filename;
```

```
  readonly attribute unsigned long lineno;
  readonly attribute unsigned long colno;
  readonly attribute any error;
};

dictionary ErrorEventInit : EventInit {
  DOMString message = "";
  USVString filename = "";
  unsigned long lineno = 0;
  unsigned long colno = 0;
  any error;
};
```

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. It must initially be initialized to undefined. Where appropriate, it is set to the object representing the error (e.g., the exception object in the case of an uncaught exception).

8.1.4.7 Unhandled promise rejections §p1169

In addition to synchronous runtime script errors[p1167], scripts may experience asynchronous promise rejections, tracked via the unhandledrejection[p1576] and rejectionhandled[p1576] events. Tracking these rejections is done via the HostPromiseRejectionTracker[p1185] abstract operation, but reporting them is defined here.

To **notify about rejected promises** given a global object[p1145] *global*:

1. Let *list* be a clone of *global*'s about-to-be-notified rejected promises list[p1145].
2. If *list* is empty, then return.
3. Empty *global*'s about-to-be-notified rejected promises list[p1145].
4. Queue a global task[p1195] on the DOM manipulation task source[p1203] given *global* to run the following step:
    1. For each promise *p* of *list*:
        1. If *p*.[[PromiseIsHandled]] is true, then continue.
        2. Let *notCanceled* be the result of firing an event named unhandledrejection[p1576] at *global*, using PromiseRejectionEvent[p1169], with the cancelable attribute initialized to true, the promise[p1170] attribute initialized to *p*, and the reason[p1170] attribute initialized to *p*.[[PromiseResult]].
        3. If *notCanceled* is true, then the user agent may report *p*.[[PromiseResult]] to a developer console.
        4. If *p*.[[PromiseIsHandled]] is false, then append *p* to *global*'s outstanding rejected promises weak set[p1145].

The PromiseRejectionEvent[p1169] interface is defined as follows:

IDL
```
[Exposed=*]
interface PromiseRejectionEvent : Event {
  constructor(DOMString type, PromiseRejectionEventInit eventInitDict);

  readonly attribute object promise;
  readonly attribute any reason;
};
```

```
dictionary PromiseRejectionEventInit : EventInit {
  required object promise;
  any reason;
};
```

The **promise** attribute must return the value it was initialized to. It represents the promise which this notification is about.

Note

Because of how Web IDL conversion rules for Promise<T> types always wrap the input into a new promise, the promise attribute is of type object instead, which is more appropriate for representing an opaque handle to the original promise object.

The **reason** attribute must return the value it was initialized to. It represents the rejection reason for the promise.

8.1.4.8 Import map parse results

An **import map parse result** is a struct that is similar to a script, and also can be stored in a script element's result, but is not counted as a script for other purposes. It has the following items:

An *import map*

An import map or null.

An *error to rethrow*

A JavaScript value representing an error that will prevent using this import map, when non-null.

To **create an import map parse result** given a string *input* and a URL *baseURL*:

1. Let *result* be an import map parse result whose import map is null and whose error to rethrow is null.
2. Parse an import map string given *input* and *baseURL*, catching any exceptions. If this threw an exception, then set *result*'s error to rethrow to that exception. Otherwise, set *result*'s import map to the return value.
3. Return *result*.

To **register an import map** given a Window *global* and an import map parse result *result*:

1. If *result*'s error to rethrow is not null, then report an exception given by *result*'s error to rethrow for *global* and return.
2. Merge existing and new import maps, given *global* and *result*'s import map.

8.1.4.9 Speculation rules parse results

A **speculation rules parse result** is a struct that is similar to a script, and also can be stored in a script element's result, but is not counted as a script for other purposes. It has the following items:

A *speculation rule set*

A speculation rule set or null.

An *error to rethrow*

A JavaScript value representing an error that will prevent using these speculation rules, when non-null.

To **create a speculation rules parse result** given a string *input* and a Document *document*:

1. Let *result* be a speculation rules parse result whose speculation rule set is null and whose error to rethrow is null.
2. Parse a speculation rule set string given *input*, *document*, and *document*'s document base URL, catching any exceptions. If this threw an exception, then set *result*'s error to rethrow to that exception. Otherwise, set *result*'s speculation rule set to the return value.
3. Return *result*.

To **register speculation rules** given a Window[p965] *global*, a speculation rules parse result[p1170] *result*, and an optional boolean *queueErrors* (default false):

1. If *result*'s error to rethrow[p1170] is not null:
    1. If *queueErrors* is true, then queue a global task[p1195] on the DOM manipulation task source[p1203] given *global* to perform the following step:
        1. Report an exception[p1167] given by *result*'s error to rethrow[p1170] for *global*.
    2. Otherwise, report an exception[p1167] given by *result*'s error to rethrow[p1170] for *global*.
    3. Return.
2. Append *result*'s speculation rule set[p1170] to *global*'s associated Document[p966]'s speculation rule sets[p1128].
3. Consider speculative loads[p1128] for *global*'s associated Document[p966].

To **unregister speculation rules** given a Window[p965] *global* and a speculation rules parse result[p1170] *result*:

1. If *result*'s error to rethrow[p1170] is not null, then return.
2. Remove *result*'s speculation rule set[p1170] from *global*'s associated Document[p966]'s speculation rule sets[p1128].
3. Consider speculative loads[p1128] for *global*'s associated Document[p966].

To **update speculation rules** given a Window[p965] *global*, a speculation rules parse result[p1170] *oldResult*, and a speculation rules parse result[p1170] *newResult*:

1. Remove *oldResult*'s speculation rule set[p1170] from *global*'s associated Document[p966]'s speculation rule sets[p1128].
2. Register speculation rules[p1171] given *global*, *newResult*, and true.

¶[p1171] Note

When updating speculation rules, as opposed to registering them for the first time, we ensure that any error[p1575] events are queued as tasks, instead of synchronously fired. Although synchronously executing error[p1575] event handlers is OK when inserting script[p685] elements, it's best if other modifications do not cause such synchronous script execution.

8.1.5 Module specifier resolution §[p1171]

8.1.5.1 The resolution algorithm §[p1171]

The resolve a module specifier[p1171] algorithm is the primary entry point for converting module specifier strings into URLs. When no import maps[p1177] are involved, it is relatively straightforward, and reduces to resolving a URL-like module specifier[p1173].

When there is a non-empty import map[p1177] present, the behavior is more complex. It checks candidate entries from all applicable module specifier maps[p1177], from most-specific to least-specific scopes[p1177] (falling back to the top-level unscoped imports[p1177]), and from most-specific to least-specific prefixes. For each candidate, the resolve an imports match[p1172] algorithm will give one of the following results:

- Successful resolution of the specifier to a URL. Then the resolve a module specifier[p1171] algorithm will return that URL.
- Throwing an exception. Then the resolve a module specifier[p1171] algorithm will rethrow that exception, without any further fallbacks.
- Failing to resolve, without an error. In this case the outer resolve a module specifier[p1171] algorithm will move on to the next candidate.

In the end, if no successful resolution is found via any of the candidate module specifier maps[p1177], resolve a module specifier[p1171] will throw an exception. Thus the result is always either a URL or a thrown exception.

To **resolve a module specifier** given a script[p685]-or-null *referringScript* and a string *specifier*:

1. Let *settingsObject* and *baseURL* be null.

2. If *referringScript* is not null:
    1. Set *settingsObject* to *referringScript*'s settings object[p1152].
    2. Set *baseURL* to *referringScript*'s base URL[p1153].
3. Otherwise:
    1. Assert: there is a current settings object[p1151].
    2. Set *settingsObject* to the current settings object[p1151].
    3. Set *baseURL* to *settingsObject*'s API base URL[p1144].
4. Let *importMap* be an empty import map[p1177].
5. If *settingsObject*'s global object[p1145] implements `Window`[p965], then set *importMap* to *settingsObject*'s global object[p1145]'s import map[p1145].
6. Let *serializedBaseURL* be *baseURL*, serialized.
7. Let *asURL* be the result of resolving a URL-like module specifier[p1173] given *specifier* and *baseURL*.
8. Let *normalizedSpecifier* be the serialization of *asURL*, if *asURL* is non-null; otherwise, *specifier*.
9. Let *result* be a URL-or-null, initially null.
10. For each *scopePrefix* → *scopeImports* of *importMap*'s scopes[p1177]:
    1. If *scopePrefix* is *serializedBaseURL*, or if *scopePrefix* ends with U+002F (/) and *scopePrefix* is a code unit prefix of *serializedBaseURL*:
        1. Let *scopeImportsMatch* be the result of resolving an imports match[p1172] given *normalizedSpecifier*, *asURL*, and *scopeImports*.
        2. If *scopeImportsMatch* is not null, then set *result* to *scopeImportsMatch*, and break.
11. If *result* is null, set *result* to the result of resolving an imports match[p1172] given *normalizedSpecifier*, *asURL*, and *importMap*'s imports[p1177].
12. If *result* is null, set it to *asURL*.

    Note

    By this point, if result *was null,* specifier *wasn't remapped to anything by* importMap*, but it might have been able to be turned into a URL.*
13. If *result* is not null:
    1. Add module to resolved module set[p1177] given *settingsObject*, *serializedBaseURL*, *normalizedSpecifier*, and *asURL*.
    2. Return *result*.
14. Throw a `TypeError` indicating that *specifier* was a bare specifier, but was not remapped to anything by *importMap*.

To **resolve an imports match**, given a string *normalizedSpecifier*, a URL-or-null *asURL*, and a module specifier map[p1177] *specifierMap*:

1. For each *specifierKey* → *resolutionResult* of *specifierMap*:
    1. If *specifierKey* is *normalizedSpecifier*:
        1. If *resolutionResult* is null, then throw a `TypeError` indicating that resolution of *specifierKey* was blocked by a null entry.

            Note

            This will terminate the entire resolve a module specifier[p1171] algorithm, without any further fallbacks.
        2. Assert: *resolutionResult* is a URL.
        3. Return *resolutionResult*.

2. If all of the following are true:

    - *specifierKey* ends with U+002F (/);
    - *specifierKey* is a code unit prefix of *normalizedSpecifier*; and
    - either *asURL* is null, or *asURL* is special,

    then:

    1. If *resolutionResult* is null, then throw a `TypeError` indicating that the resolution of *specifierKey* was blocked by a null entry.

        Note

        This will terminate the entire resolve a module specifier[p1171] algorithm, without any further fallbacks.

    2. Assert: *resolutionResult* is a URL.
    3. Let *afterPrefix* be the portion of *normalizedSpecifier* after the initial *specifierKey* prefix.
    4. Assert: *resolutionResult*, serialized, ends with U+002F (/), as enforced during parsing[p1177].
    5. Let *url* be the result of URL parsing *afterPrefix* with *resolutionResult*.
    6. If *url* is failure, then throw a `TypeError` indicating that resolution of *normalizedSpecifier* was blocked since the *afterPrefix* portion could not be URL-parsed relative to the *resolutionResult* mapped to by the *specifierKey* prefix.

        Note

        This will terminate the entire resolve a module specifier[p1171] algorithm, without any further fallbacks.

    7. Assert: *url* is a URL.
    8. If the serialization of *resolutionResult* is not a code unit prefix of the serialization of *url*, then throw a `TypeError` indicating that the resolution of *normalizedSpecifier* was blocked due to it backtracking above its prefix *specifierKey*.

        Note

        This will terminate the entire resolve a module specifier[p1171] algorithm, without any further fallbacks.

    9. Return *url*.

2. Return null.

Note

The resolve a module specifier[p1171] algorithm will fall back to a less-specific scope, or to "`imports`", if possible.

To **resolve a URL-like module specifier**, given a string *specifier* and a URL *baseURL*:

1. If *specifier* starts with "`/`", "`./`", or "`../`":

    1. Let *url* be the result of URL parsing *specifier* with *baseURL*.
    2. If *url* is failure, then return null.

        Example

        One way this could happen is if *specifier* is "`../foo`" and *baseURL* is a `data:` URL.

    3. Return *url*.

    Note

    This includes cases where specifier *starts with* "`//`", *i.e., scheme-relative URLs. Thus,* url *might end up with a different host than* baseURL.

2. Let *url* be the result of URL parsing *specifier* (with no base URL).

3. If *url* is failure, then return null.

4. Return *url*.

8.1.5.2 Import maps §p11 74

An import map[p1177] allows control over module specifier resolution. Import maps are delivered via inline `script`[p685] elements with their `type`[p686] attribute set to "`importmap`", and with their child text content containing a JSON representation of the import map.

A `Document`[p135] can have multiple import maps processed, which can happen either before or after any modules have been imported, e.g., via `import()` expressions or `script`[p685] elements with their `type`[p686] attribute set to "`module`". The merge existing and new import maps[p1178] algorithm ensures that new import maps cannot define the module resolution for modules that were already defined by past import maps, or for ones that were already resolved.

¶p11 74 **Example**

The simplest use of import maps is to globally remap a bare module specifier:

```
{
  "imports": {
    "moment": "/node_modules/moment/src/moment.js"
  }
}
```

This enables statements like `import moment from "moment";` to work, fetching and evaluating the JavaScript module at the `/node_modules/moment/src/moment.js` URL.

¶p11 74 **Example**

An import map can remap a class of module specifiers into a class of URLs by using trailing slashes, like so:

```
{
  "imports": {
    "moment/": "/node_modules/moment/src/"
  }
}
```

This enables statements like `import localeData from "moment/locale/zh-cn.js";` to work, fetching and evaluating the JavaScript module at the `/node_modules/moment/src/locale/zh-cn.js` URL. Such trailing-slash mappings are often combined with bare-specifier mappings, e.g.

```
{
  "imports": {
    "moment": "/node_modules/moment/src/moment.js",
    "moment/": "/node_modules/moment/src/"
  }
}
```

so that both the "main module" specified by "`moment`" and the "submodules" specified by paths such as "`moment/locale/zh-cn.js`" are available.

¶p11 74 **Example**

Bare specifiers are not the only type of module specifiers which import maps can remap. "URL-like" specifiers, i.e., those that are either parseable as absolute URLs or start with "`/`", "`./`", or "`../`", can be remapped as well:

```
{
  "imports": {
    "https://cdn.example.com/vue/dist/vue.runtime.esm.js": "/node_modules/vue/dist/
vue.runtime.esm.js",
```

```
    "/js/app.mjs": "/js/app-8e0d62a03.mjs",
    "../helpers/": "https://cdn.example/helpers/"
  }
}
```

Note how the URL to be remapped, as well as the URL being mapped to, can be specified either as absolute URLs, or as relative URLs starting with "`/`", "`./`", or "`../`". (They cannot be specified as relative URLs without those starting sigils, as those help distinguish from bare module specifiers.) Also note how the trailing slash mapping[p1174] works in this context as well.

Such remappings operate on the post-canonicalization URL, and do not require a match between the literal strings supplied in the import map key and the imported module specifier. So for example, if this import map was included on `https://example.com/app.html`, then not only would `import "/js/app.mjs"` be remapped, but so would `import "./js/app.mjs"` and `import "./foo/../js/app.mjs"`.

¶[p1175] **Example**

All previous examples have globally remapped module specifiers, by using the top-level "`imports`" key in the import map. The top-level "`scopes`" key can be used to provide localized remappings, which only apply when the referring module matches a specific URL prefix. For example:

```
{
  "scopes": {
    "/a/" : {
      "moment": "/node_modules/moment/src/moment.js"
    },
    "/b/" : {
      "moment": "https://cdn.example.com/moment/src/moment.js"
    }
  }
}
```

With this import map, the statement `import "moment"` will have different meanings depending on which referrer script contains the statement:

- Inside scripts located under `/a/`, this will import `/node_modules/moment/src/moment.js`.
- Inside scripts located under `/b/`, this will import `https://cdn.example.com/moment/src/moment.js`.
- Inside scripts located under `/c/`, this will fail to resolve and thus throw an exception.

A typical usage of scopes is to allow multiple versions of the "same" module to exist in a web application, with some parts of the module graph importing one version, and other parts importing another version.

¶[p1176] **Example**

Scopes can overlap each other, and overlap the global "`imports`" specifier map. At resolution time, scopes are consulted in order of most- to least-specific, where specificity is measured by sorting the scopes using the code unit less than operation. So, for example, "`/scope2/scope3/`" is treated as more specific than "`/scope2/`", which is treated as more specific than the top-level (unscoped) mappings.

The following import map illustrates this:

```
{
  "imports": {
    "a": "/a-1.mjs",
    "b": "/b-1.mjs",
    "c": "/c-1.mjs"
  },
  "scopes": {
    "/scope2/": {
      "a": "/a-2.mjs"
```

```
    },
    "/scope2/scope3/": {
      "b": "/b-3.mjs"
    }
  }
}
```

This results in the following resolutions (using relative URLs for brevity):

		Specifier		
		"a"	"b"	"c"
Referrer	**/scope1/r.mjs**	/a-1.mjs	/b-1.mjs	/c-1.mjs
	/scope2/r.mjs	/a-2.mjs	/b-1.mjs	/c-1.mjs
	/scope2/scope3/r.mjs	/a-2.mjs	/b-3.mjs	/c-1.mjs

¶ p11 76 **Example**

Import maps can also be used to provide modules with integrity metadata to be used in *Subresource Integrity* checks. [SRI] p1586

The following import map illustrates this:

```
{
  "imports": {
    "a": "/a-1.mjs",
    "b": "/b-1.mjs",
    "c": "/c-1.mjs"
  },
  "integrity": {
    "/a-1.mjs": "sha384-Li9vy3DqF8tnTXuiaAJuML3ky+er10rcgNR/VqsVpcw+ThHmYcwiB1pbOxEbzJr7",
    "/d-1.mjs": "sha384-MBO5IDfYaE6c6Aao94oZrIOiC6CGiSN2n4QUbHNPhzk5Xhm0djZLQqTpL0HzTUxk"
  }
}
```

The above example provides integrity metadata to be enforced on the modules `/a-1.mjs` and `/d-1.mjs`, even if the latter is not defined as an import in the map.

The child text content of a `script` p685 element representing an import map p1177 must match the following **import map authoring requirements**:

- It must be valid JSON. [JSON] p1583
- The JSON must represent a JSON object, with at most the three keys "`imports`", "`scopes`", and "`integrity`".
- The values corresponding to the "`imports`", "`scopes`", and "`integrity`" keys, if present, must themselves be JSON objects.
- The value corresponding to the "`imports`" key, if present, must be a valid module specifier map p1176.
- The value corresponding to the "`scopes`" key, if present, must be a JSON object, whose keys are valid URL strings and whose values are valid module specifier maps p1176.
- The value corresponding to the "`integrity`" key, if present, must be a JSON object, whose keys are valid URL strings and whose values fit the requirements of the integrity attribute.

A **valid module specifier map** is a JSON object that meets the following requirements:

- All of its keys must be nonempty.
- All of its values must be strings.
- Each value must be either a valid absolute URL or a valid URL string that starts with "`/`", "`./`", or "`../`".
- If a given key ends with "`/`", then the corresponding value must also.

8.1.5.3 Import map processing model §[p1177]

Formally, an **import map** is a struct with three items:

- **imports**, a module specifier map[p1177];
- **scopes**, an ordered map of URLs to module specifier maps[p1177]; and
- **integrity**, a module integrity map[p1177].

A **module specifier map** is an ordered map whose keys are strings and whose values are either URLs or nulls.

A **module integrity map** is an ordered map whose keys are URLs and whose values are strings that will be used as integrity metadata.

An **empty import map** is an import map[p1177] with its imports[p1177] and scopes[p1177] both being empty maps.

A **specifier resolution record** is a struct. It has the following items:

A *serialized base URL*
: A string-or-null that represents the base URL of the specifier, when one exists.

A *specifier*
: A string representing the specifier.

A *specifier as a URL*
: A URL-or-null that represents the URL in case of a URL-like module specifier.

Note

Implementations can replace specifier as a URL[p1177] with a boolean that indicates that the specifier is either bare or URL-like that is special.

To **add module to resolved module set** given an environment settings object[p1144] *settingsObject*, a string *serializedBaseURL*, a string *normalizedSpecifier*, and a URL-or-null *asURL*:

1. Let *global* be *settingsObject*'s global object[p1145].
2. If *global* does not implement `Window`[p965], then return.
3. Let *record* be a new specifier resolution record[p1177], with serialized base URL[p1177] set to *serializedBaseURL*, specifier[p1177] set to *normalizedSpecifier*, and specifier as a URL[p1177] set to *asURL*.
4. Append *record* to *global*'s resolved module set[p1145].

To **parse an import map string**, given a string *input* and a URL *baseURL*:

1. Let *parsed* be the result of parsing a JSON string to an Infra value given *input*.
2. If *parsed* is not an ordered map, then throw a `TypeError` indicating that the top-level value needs to be a JSON object.
3. Let *sortedAndNormalizedImports* be an empty ordered map.
4. If *parsed*["`imports`"] exists:
    1. If *parsed*["`imports`"] is not an ordered map, then throw a `TypeError` indicating that the value for the "`imports`" top-level key needs to be a JSON object.
    2. Set *sortedAndNormalizedImports* to the result of sorting and normalizing a module specifier map[p1182] given *parsed*["`imports`"] and *baseURL*.
5. Let *sortedAndNormalizedScopes* be an empty ordered map.
6. If *parsed*["`scopes`"] exists:
    1. If *parsed*["`scopes`"] is not an ordered map, then throw a `TypeError` indicating that the value for the "`scopes`" top-

level key needs to be a JSON object.

2. Set *sortedAndNormalizedScopes* to the result of sorting and normalizing scopes[p1183] given *parsed*["scopes"] and *baseURL*.

7. Let *normalizedIntegrity* be an empty ordered map.

8. If *parsed*["integrity"] exists:

    1. If *parsed*["integrity"] is not an ordered map, then throw a TypeError indicating that the value for the "integrity" top-level key needs to be a JSON object.

    2. Set *normalizedIntegrity* to the result of normalizing a module integrity map[p1183] given *parsed*["integrity"] and *baseURL*.

9. If *parsed*'s keys contains any items besides "imports", "scopes", or "integrity", then the user agent should report a warning to the console indicating that an invalid top-level key was present in the import map.

    Note

    This can help detect typos. It is not an error, because that would prevent any future extensions from being added backward-compatibly.

10. Return an import map[p1177] whose imports[p1177] are *sortedAndNormalizedImports*, whose scopes[p1177] are *sortedAndNormalizedScopes*, and whose integrity[p1177] are *normalizedIntegrity*.

¶ p1178

Example

The import map[p1177] that results from this parsing algorithm is highly normalized. For example, given a base URL of `https://example.com/base/page.html`, the input

```
{
  "imports": {
    "/app/helper": "node_modules/helper/index.mjs",
    "lodash": "/node_modules/lodash-es/lodash.js"
  }
}
```

will generate an import map[p1177] with imports[p1177] of

```
«[
  "https://example.com/app/helper" → https://example.com/base/node_modules/helper/index.mjs
  "lodash" → https://example.com/node_modules/lodash-es/lodash.js
]»
```

and (despite nothing being present in the input string) an empty ordered map for its scopes[p1177].

To **merge module specifier maps**, given a module specifier map[p1177] *newMap* and a module specifier map[p1177] *oldMap*:

1. Let *mergedMap* be a deep copy of *oldMap*.

2. For each *specifier* → *url* of *newMap*:

    1. If *specifier* exists in *oldMap*:

        1. The user agent may report a warning to the console indicating the ignored rule. They may choose to avoid reporting if the rule is identical to an existing one.

        2. Continue.

    2. Set *mergedMap*[*specifier*] to *url*.

3. Return *mergedMap*.

To **merge existing and new import maps**, given a global object[p1145] *global* and an import map[p1177] *newImportMap*:

1. Let *newImportMapScopes* be a deep copy of *newImportMap*'s scopes[p1177].

   Note

   We're mutating these copies and removing items from them when they are used to ignore scope-specific rules. This is true for newImportMapScopes*, as well as to* newImportMapImports *below.*

2. Let *oldImportMap* be *global*'s import map[p1145].

3. Let *newImportMapImports* be a deep copy of *newImportMap*'s imports[p1177].

4. For each *scopePrefix* → *scopeImports* of *newImportMapScopes*:

   1. For each *record* of *global*'s resolved module set[p1145]:

      1. If *scopePrefix* is *record*'s serialized base URL[p1177], or if *scopePrefix* ends with U+002F (/) and *scopePrefix* is a code unit prefix of *record*'s serialized base URL[p1177]:

         1. For each *specifierKey* → *resolutionResult* of *scopeImports*:

            1. If *specifierKey* is *record*'s specifier[p1177], or if all of the following conditions are true:

               - *specifierKey* ends with U+002F (/);
               - *specifierKey* is a code unit prefix of *record*'s specifier[p1177];
               - either *record*'s specifier as a URL[p1177] is null or is special,

               then:

               1. The user agent may report a warning to the console indicating the ignored rule. They may choose to avoid reporting if the rule is identical to an existing one.
               2. Remove *scopeImports*[*specifierKey*].

      Note

      Implementers are encouraged to implement a more efficient matching algorithm when working with the resolved module set[p1145]. As guidance, the number of resolved/mapped modules in a large application can be on the order of thousands.

      2. If *scopePrefix* exists in *oldImportMap*'s scopes[p1177], then set *oldImportMap*'s scopes[p1177][*scopePrefix*] to the result of merging module specifier maps[p1178], given *scopeImports* and *oldImportMap*'s scopes[p1177][*scopePrefix*].

      3. Otherwise, set *oldImportMap*'s scopes[p1177][*scopePrefix*] to *scopeImports*.

5. For each *url* → *integrity* of *newImportMap*'s integrity[p1177]:

   1. If *url* exists in *oldImportMap*'s integrity[p1177]:

      1. The user agent may report a warning to the console indicating the ignored rule. They may choose to avoid reporting if the rule is identical to an existing one.
      2. Continue.

   2. Set *oldImportMap*'s integrity[p1177][*url*] to *integrity*.

6. For each *record* of *global*'s resolved module set[p1145]:

   1. For each *specifier* → *url* of *newImportMapImports*:

      1. If *specifier* starts with *record*'s specifier[p1177]:

         1. The user agent may report a warning to the console indicating the ignored rule. They may choose to avoid reporting if the rule is identical to an existing one.
         2. Remove *newImportMapImports*[*specifier*].

7. Set *oldImportMap*'s imports[p1177] to the result of merge module specifier maps[p1178], given *newImportMapImports* and *oldImportMap*'s imports[p1177].

The above algorithm merges a new import map into the given environment settings object[p1144]'s global object[p1145]'s import map[p1177]. Let's examine a few examples:

¶[p1180] Example

There are two cases when rules of the new import map don't get merged into the existing one.

1. The new import map rule has the exact same scope and specifier as a rule in the existing import map. We'll call that "conflicting rule".
2. The new import map rule may impact the resolution of an already resolved module. We'll call that "impacted already resolved module".

When the new import map has no conflicting rules, and there are no impacted resolved modules, the resulting map would be a combination of the new and existing maps. Rules that would have individually impacted similar modules (e.g. "/app/" and "/app/helper") but are not an exact match are not conflicting, and all make it to the merged map.

So, the following existing and new import maps:

```
{
   "imports": {
    "/app/": "./original-app/",
  }
}
```

```
{
  "imports": {
    "/app/helper": "./helper/index.mjs"
  },
  "scopes": {
    "/js": {
      "/app/": "./js-app/"
    }
  }
}
```

Would be equivalent to the following single import map:

```
{
  "imports": {
    "/app/": "./original-app/",
    "/app/helper": "./helper/index.mjs"
  },
  "scopes": {
    "/js": {
      "/app/": "./js-app/"
    }
  }
}
```

¶[p1180] Example

When the new import map impacts an already resolved module, that rule gets dropped from the import map.

So, if the resolved module set[p1145] already contains the "`/app/helper`", the following new import map:

```
{
   "imports": {
    "/app/helper": "./helper/index.mjs",
    "lodash": "/node_modules/lodash-es/lodash.js"
  }
}
```

Would be equivalent to the following one:

```
{
  "imports": {
    "lodash": "/node_modules/lodash-es/lodash.js"
  }
}
```

¶ p11 81 **Example**

The same is true for rules that impact already resolved modules defined in specific scopes. If we already resolved "`/app/helper`" from "`/app/main.mjs`" the following new import map:

```
{
  "scopes": {
    "/app/": {
      "/app/helper": "./helper/index.mjs"
    }
  },
   "imports": {
    "lodash": "/node_modules/lodash-es/lodash.js"
  }
}
```

Would similarly be equivalent to:

```
{
  "imports": {
    "lodash": "/node_modules/lodash-es/lodash.js"
  }
}
```

¶ p11 81 **Example**

We could also have cases where a single already-resolved module specifier has multiple rules for its resolution, depending on the referring script. In such cases, only the relevant rules would not be added to the map.

For example, if we already resolved "`/app/helper`" from "`/app/vendor/main.mjs`", the following new import map:

```
{
  "scopes": {
    "/app/": {
      "/app/helper": "./helper/index.mjs"
    },
    "/app/vendor/": {
      "/app/": "./vendor_helper/"
    },
    "/vendor/": {
      "/app/helper": "./helper/vendor_index.mjs"
    }
  },
   "imports": {
    "lodash": "/node_modules/lodash-es/lodash.js"
    "/app/": "./general_app_path/"
    "/app/helper": "./other_path/helper/index.mjs"
  }
}
```

Would be equivalent to:

```
{
  "scopes": {
    "/vendor/": {
      "/app/helper": "./helper/vendor_index.mjs"
    }
  },
  "imports": {
    "lodash": "/node_modules/lodash-es/lodash.js"
  }
}
```

This is achieved by the fact that the merge algorithm tracks already resolved modules and removes rules affecting them from new import maps before they are merged into the existing one.

¶ p11 82 **Example**

When the new import map has conflicting rules to the existing import map, with no impacted already resolved modules, the existing import map rules persist.

For example, the following existing and new import maps:

```
{
   "imports": {
    "/app/helper": "./helper/index.mjs",
    "lodash": "/node_modules/lodash-es/lodash.js"
  }
}
```

```
{
  "imports": {
    "/app/helper": "./main/helper/index.mjs"
  }
}
```

Would be equivalent to the following single import map:

```
{
  "imports": {
    "/app/helper": "./helper/index.mjs",
    "lodash": "/node_modules/lodash-es/lodash.js",
  }
}
```

To **sort and normalize a module specifier map**, given an ordered map *originalMap* and a URL *baseURL*:

1. Let *normalized* be an empty ordered map.
2. For each *specifierKey* → *value* of *originalMap*:
    1. Let *normalizedSpecifierKey* be the result of normalizing a specifier key[p1184] given *specifierKey* and *baseURL*.
    2. If *normalizedSpecifierKey* is null, then continue.
    3. If *value* is not a string:
        1. The user agent may report a warning to the console indicating that addresses need to be strings.
        2. Set *normalized*[*normalizedSpecifierKey*] to null.
        3. Continue.

4. Let *addressURL* be the result of resolving a URL-like module specifier[p1173] given *value* and *baseURL*.
5. If *addressURL* is null:
    1. The user agent may report a warning to the console indicating that the address was invalid.
    2. Set *normalized*[*normalizedSpecifierKey*] to null.
    3. Continue.
6. If *specifierKey* ends with U+002F (/), and the serialization of *addressURL* does not end with U+002F (/):
    1. The user agent may report a warning to the console indicating that an invalid address was given for the specifier key *specifierKey*; since *specifierKey* ends with a slash, the address needs to as well.
    2. Set *normalized*[*normalizedSpecifierKey*] to null.
    3. Continue.
7. Set *normalized*[*normalizedSpecifierKey*] to *addressURL*.

3. Return the result of sorting in descending order *normalized*, with an entry *a* being less than an entry *b* if *a*'s key is code unit less than *b*'s key.

To **sort and normalize scopes**, given an ordered map *originalMap* and a URL *baseURL*:

1. Let *normalized* be an empty ordered map.
2. For each *scopePrefix* → *potentialSpecifierMap* of *originalMap*:
    1. If *potentialSpecifierMap* is not an ordered map, then throw a `TypeError` indicating that the value of the scope with prefix *scopePrefix* needs to be a JSON object.
    2. Let *scopePrefixURL* be the result of URL parsing *scopePrefix* with *baseURL*.
    3. If *scopePrefixURL* is failure:
        1. The user agent may report a warning to the console that the scope prefix URL was not parseable.
        2. Continue.
    4. Let *normalizedScopePrefix* be the serialization of *scopePrefixURL*.
    5. Set *normalized*[*normalizedScopePrefix*] to the result of sorting and normalizing a module specifier map[p1182] given *potentialSpecifierMap* and *baseURL*.
3. Return the result of sorting in descending order *normalized*, with an entry *a* being less than an entry *b* if *a*'s key is code unit less than *b*'s key.

Note

In the above two algorithms, sorting keys and scopes in descending order has the effect of putting `"foo/bar/"` *before* `"foo/"`*. This in turn gives* `"foo/bar/"` *a higher priority than* `"foo/"` *during module specifier resolution*[p1171]*.*

To **normalize a module integrity map**, given an ordered map *originalMap*:

1. Let *normalized* be an empty ordered map.
2. For each *key* → *value* of *originalMap*:
    1. Let *resolvedURL* be the result of resolving a URL-like module specifier[p1173] given *key* and *baseURL*.

        Note

        Unlike `"imports"`*, keys of the integrity map are treated as URLs, not module specifiers. However, we use the resolve a URL-like module specifier*[p1173] *algorithm to prohibit "bare" relative URLs like* `foo`*, which could be mistaken for module specifiers.*

    2. If *resolvedURL* is null:
        1. The user agent may report a warning to the console indicating that the key failed to resolve.

2. Continue.

3. If *value* is not a string:

    1. The user agent may report a warning to the console indicating that integrity metadata values need to be strings.

    2. Continue.

4. Set *normalized*[*resolvedURL*] to *value*.

3. Return *normalized*.

To **normalize a specifier key**, given a string *specifierKey* and a URL *baseURL*:

1. If *specifierKey* is the empty string:

    1. The user agent may report a warning to the console indicating that specifier keys may not be the empty string.

    2. Return null.

2. Let *url* be the result of resolving a URL-like module specifier[p1173], given *specifierKey* and *baseURL*.

3. If *url* is not null, then return the serialization of *url*.

4. Return *specifierKey*.

8.1.6 JavaScript specification host hooks §[p11 84]

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.

8.1.6.1 HostEnsureCanAddPrivateElement(*O*) §[p11 84]

JavaScript contains an implementation-defined HostEnsureCanAddPrivateElement(*O*) abstract operation. User agents must use the following implementation: [JAVASCRIPT][p1583]

1. If *O* is a `WindowProxy`[p977] object, or implements `Location`[p980], then return ThrowCompletion(a new `TypeError`).

2. Return NormalCompletion(unused).

Note

JavaScript private fields can be applied to arbitrary objects. Since this can dramatically complicate implementation for particularly-exotic host objects, the JavaScript language specification provides this hook to allow hosts to reject private fields on objects meeting a host-defined criteria. In the case of HTML, `WindowProxy`[p977] and `Location`[p980] have complicated semantics — particularly around navigation and security — that make implementation of private field semantics challenging, so our implementation simply rejects those objects.

8.1.6.2 HostEnsureCanCompileStrings(*realm*, *parameterStrings*, *bodyString*, *codeString*, *compilationType*, *parameterArgs*, *bodyArg*) §[p11 84]

JavaScript contains an implementation-defined HostEnsureCanCompileStrings abstract operation, redefined by the *Dynamic Code Brand Checks* proposal. User agents must use the following implementation: [JAVASCRIPT][p1583] [JSDYNAMICCODEBRANDCHECKS][p1583]

1. Perform ? EnsureCSPDoesNotBlockStringCompilation(*realm*, *parameterStrings*, *bodyString*, *codeString*, *compilationType*, *parameterArgs*, *bodyArg*). [CSP][p1580]

8.1.6.3 HostGetCodeForEval(*argument*) §[p11 85]

The *Dynamic Code Brand Checks* proposal contains an implementation-defined HostGetCodeForEval(*argument*) abstract operation. User agents must use the following implementation: [JSDYNAMICCODEBRANDCHECKS][p1583]

1. If *argument* is a `TrustedScript` object, then return *argument*'s data.
2. Otherwise, return no-code.

8.1.6.4 HostPromiseRejectionTracker(*promise*, *operation*) §[p11 85]

JavaScript contains an implementation-defined HostPromiseRejectionTracker(*promise*, *operation*) abstract operation. User agents must use the following implementation: [JAVASCRIPT][p1583]

1. Let *script* be the running script[p1166].
2. If *script* is a classic script[p1153] and *script*'s muted errors[p1153] is true, then return.
3. Let *settingsObject* be the current settings object[p1151].
4. If *script* is not null, then set *settingsObject* to *script*'s settings object[p1152].
5. Let *global* be *settingsObject*'s global object[p1145].
6. If *operation* is "`reject`":
    1. Append *promise* to *global*'s about-to-be-notified rejected promises list[p1145].
7. If *operation* is "`handle`":
    1. If *global*'s about-to-be-notified rejected promises list[p1145] contains *promise*, then remove *promise* from that list and return.
    2. If *global*'s outstanding rejected promises weak set[p1145] does not contain *promise*, then return.
    3. Remove *promise* from *global*'s outstanding rejected promises weak set[p1145].
    4. Queue a global task[p1195] on the DOM manipulation task source[p1203] given *global* to fire an event named `rejectionhandled`[p1576] at *global*, using `PromiseRejectionEvent`[p1169], with the `promise`[p1170] attribute initialized to *promise*, and the `reason`[p1170] attribute initialized to *promise*.[[PromiseResult]].

8.1.6.5 HostSystemUTCEpochNanoseconds(*global*) §[p11 85]

The Temporal proposal contains an implementation-defined HostSystemUTCEpochNanoseconds abstract operation. User agents must use the following implementation: [JSTEMPORAL][p1583]

1. Let *settingsObject* be *global*'s relevant settings object[p1151].
2. Let *time* be *settingsObject*'s current wall time.
3. Let *ns* be the number of nanoseconds from the Unix epoch to *time*, rounded to the nearest integer.
4. Return the result of clamping *ns* between nsMinInstant and nsMaxInstant.

8.1.6.6 Job-related host hooks §[p11 85]

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. HTML uses these abstract operations to track the incumbent settings object[p1149] in promises and `FinalizationRegistry` callbacks by saving and restoring the incumbent settings object[p1149] and a JavaScript execution context for the active script[p1154] in JobCallbacks. This section defines them for user agent hosts.

8.1.6.6.1 HostCallJobCallback(callback, V, argumentsList) §p11 86

JavaScript contains an implementation-defined HostCallJobCallback(*callback*, *V*, *argumentsList*) abstract operation to let hosts restore state when invoking JavaScript callbacks from inside tasks. User agents must use the following implementation: [JAVASCRIPT]p1583

1. Let *incumbent settings* be *callback*.[[HostDefined]].[[IncumbentSettings]].
2. Let *script execution context* be *callback*.[[HostDefined]].[[ActiveScriptContext]].
3. Prepare to run a callbackp1148 with *incumbent settings*.

   Note

   This affects the incumbentp1146 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 scriptp1154 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 callbackp1148 with *incumbent settings*.
8. Return *result*.

8.1.6.6.2 HostEnqueueFinalizationRegistryCleanupJob(finalizationRegistry) §p11 86

JavaScript has the ability to register objects with `FinalizationRegistry` objects, in order to schedule a cleanup action if they are found to be garbage collected. The JavaScript specification contains an implementation-defined HostEnqueueFinalizationRegistryCleanupJob(*finalizationRegistry*) abstract operation to schedule the cleanup action.

Note

The timing and occurrence of cleanup work is implementation-defined in the JavaScript specification. User agents might differ in when and whether an object is garbage collected, affecting both whether the return value of the `WeakRef.prototype.deref()` method is undefined, and whether `FinalizationRegistry` cleanup callbacks occur. There are well-known cases in popular web browsers where objects are not accessible to JavaScript, but they remain retained by the garbage collector indefinitely. HTML clears kept-alive `WeakRef` objects in the perform a microtask checkpointp1200 algorithm. Authors would be best off not depending on the timing details of garbage collection implementations.

Cleanup actions do not take place interspersed with synchronous JavaScript execution, but rather happen in queued tasksp1194. User agents must use the following implementation: [JAVASCRIPT]p1583

1. Let *global* be *finalizationRegistry*.[[Realm]]'s global objectp1145.
2. Queue a global taskp1195 on the **JavaScript engine task source** given *global* to perform the following steps:
   1. Let *entry* be *finalizationRegistry*.[[CleanupCallback]].[[Callback]].[[Realm]]'s environment settings objectp1145.
   2. Prepare to run scriptp1166 with *entry*.

      Note

      This affects the entryp1146 concept while the cleanup callback runs.

   3. Let *result* be the result of performing CleanupFinalizationRegistry(*finalizationRegistry*).
   4. Clean up after running scriptp1166 with *entry*.
   5. If *result* is an abrupt completion, then report an exceptionp1167 given by *result*.[[Value]] for *global*.

8.1.6.6.3 HostEnqueueGenericJob(job, realm) §[p11 87]

JavaScript contains an implementation-defined HostEnqueueGenericJob(*job*, *realm*) abstract operation to perform generic jobs in a particular realm (e.g., resolve promises resulting from `Atomics.waitAsync`). User agents must use the following implementation: [JAVASCRIPT][p1583]

1. Let *global* be *realm*'s global object[p1145].
2. Queue a global task[p1195] on the JavaScript engine task source[p1186] given *global* to perform *job*().

8.1.6.6.4 HostEnqueuePromiseJob(job, realm) §[p11 87]

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][p1583]

1. If *realm* is not null, then let *job settings* be the settings object[p1145] 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.*

   NewPromiseResolveThenableJob and NewPromiseReactionJob both seem to provide non-null realms (the current Realm Record) in the case of a revoked proxy. The previous text could be updated to reflect that.

2. Queue a microtask[p1195] to perform the following steps:

   1. If *job settings* is not null, then prepare to run script[p1166] with *job settings*.

      Note

      This affects the entry[p1146] concept while the job runs.

   2. Let *result* be *job*().

      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[p1149] and a JavaScript execution context for to the active script[p1154] in HostMakeJobCallback[p1188] and restores them in HostCallJobCallback[p1186].*

   3. If *job settings* is not null, then clean up after running script[p1166] with *job settings*.

   4. If *result* is an abrupt completion, then report an exception[p1167] given by *result*.[[Value]] for *realm*'s global object[p1145].

      There is a very gnarly case where HostEnqueuePromiseJob is called with a null realm (e.g., because Promise.prototype.then was called with null handlers) but also the job returns abruptly (because the promise capability's resolve or reject handler threw, possibly because this is a subclass of Promise that takes the supplied functions and wraps them in throwing functions before passing them on to the function passed to the Promise superclass constructor). Which global is to be used then, considering that the current realm could be different at each of those steps, by using a Promise constructor or Promise.prototype.then from another realm? See issue #10526.

8.1.6.6.5 HostEnqueueTimeoutJob(job, realm, milliseconds) §p11 88

JavaScript contains an implementation-defined HostEnqueueTimeoutJob(*job*, *milliseconds*) abstract operation to schedule an operation to be performed after a timeout. HTML schedules these operations using run steps after a timeout[p1256]. User agents must use the following implementation: [JAVASCRIPT][p1583]

1. Let *global* be *realm*'s global object[p1145].
2. Let *timeoutStep* be an algorithm step which queues a global task[p1195] on the JavaScript engine task source[p1186] given *global* to perform *job*().
3. Run steps after a timeout[p1256] given *global*, "`JavaScript`", *milliseconds*, and *timeoutStep*.

8.1.6.6.6 HostMakeJobCallback(callable) §p11 88

JavaScript contains an implementation-defined HostMakeJobCallback(*callable*) abstract operation to let hosts attach state to JavaScript callbacks that are called from inside task[p1194]s. User agents must use the following implementation: [JAVASCRIPT][p1583]

1. Let *incumbent settings* be the incumbent settings object[p1149].
2. Let *active script* be the active script[p1154].
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[p1152]'s realm[p1145], and its ScriptOrModule set to *active script*'s record[p1152].

   Note

   As seen below, this is used in order to propagate the current active script[p1154] 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:

   ```
   Promise.resolve('import(`./example.mjs`)').then(eval);
   ```

   Without this step (and the steps that use it in HostCallJobCallback[p1186]), there would be no active script[p1154] when the `import()` expression is evaluated, since `eval()` is a built-in function that does not originate from any particular script[p1152].

   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[p1153] appropriately.

   Example

   active script can be null if the user clicks on the following button:

   ```
   <button onclick="Promise.resolve('import(`./example.mjs`)').then(eval)">Click me</button>
   ```

   In this case, the JavaScript function for the event handler[p1206] will be created by the get the current value of the event handler[p1211] algorithm, which creates a function with null [[ScriptOrModule]] value. Thus, when the promise machinery calls HostMakeJobCallback[p1188], there will be no active script[p1154] to pass along.

   As a consequence, this means that when the `import()` expression is evaluated, there will still be no active script[p1154]. Fortunately that is handled by our implementation of HostLoadImportedModule[p1191] by falling back to using the current settings object[p1151]'s API base URL[p1144].

5. Return the JobCallback Record { [[Callback]]: *callable*, [[HostDefined]]: { [[IncumbentSettings]]: *incumbent settings*, [[ActiveScriptContext]]: *script execution context* } }.

8.1.6.7 Module-related host hooks §p1189

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`p685 element with `type`p686 attribute set to "`module`", and how modules are fetched, resolved, and executed. [JAVASCRIPT]p1583

Note

Although the JavaScript specification speaks in terms of "scripts" versus "modules", in general this specification speaks in terms of classic scriptsp1153 versus module scriptsp1153, since both of them use the `script`p685 element.

For web developers (non-normative)

***modulePromise* = `import(`*specifier*`)`**

Returns a promise for the module namespace object for the module scriptp1153 identified by *specifier*. This allows dynamic importing of module scripts at runtime, instead of statically using the `import` statement form. The specifier will be resolvedp1171 relative to the active scriptp1154.

The returned promise will be rejected if an invalid specifier is given, or if a failure is encountered while fetchingp1191 or evaluating the resulting module graph.

This syntax can be used inside both classicp1153 and module scriptsp1153. It thus provides a bridge into the module-script world, from the classic-script world.

***url* = `import.meta.url`p1190**

Returns the active module scriptp1154's base URLp1153.

This syntax can only be used inside module scriptsp1153.

***url* = `import.meta.resolve`p1190`(`*specifier*`)`**

Returns *specifier*, resolvedp1171 relative to the active scriptp1154. That is, this returns the URL that would be imported by using `import(`*specifier*`)`.

Throws a `TypeError` exception if an invalid specifier is given.

This syntax can only be used inside module scriptsp1153.

A **module map** is a map keyed by tuples consisting of a URL record and a string. The URL record is the request URL at which the module was fetched, and the string indicates the type of the module (e.g. "`javascript-or-wasm`"). The module mapp1189's values are either a module scriptp1153 or a list of algorithms (that are waiting for the fetch to complete). Module mapsp1189 are used to ensure that imported module scripts are only fetched, parsed, and evaluated once per `Document`p135 or workerp1306.

Example

Since module mapsp1189 are keyed by (URL, module type), the following code will create three separate entries in the module mapp1189, since it results in three different (URL, module type) tuples (all with "`javascript-or-wasm`" type):

```
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 mapp1189; 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 mapp1189, since after applying the URL parser to these inputs, the resulting URL records are equal:

```
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 workersp1333 are keyed by their parsed constructor URLp1324.

Example

Since module type is also part of the module map[p1189] key, the following code will create two separate entries in the module map[p1189] (the type is "javascript-or-wasm" for the first, and "css" for the second):

```
<script type=module>
  import "https://example.com/module";
</script>
<script type=module>
  import "https://example.com/module" with { type: "css" };
</script>
```

This can result in two separate fetches and two separate module evaluations being performed.

In practice, due to the as-yet-unspecified memory cache (see issue #6110) the resource may only be fetched once in WebKit and Blink-based browsers. Additionally, as long as all module types are mutually exclusive, the module type check in fetch a single module script[p1160] will fail for at least one of the imports, so at most one module evaluation will occur.

The purpose of including the type in the module map[p1189] key is so that an import with the wrong type attribute does not prevent a different import of the same specifier but with the correct type from succeeding.

Example

JavaScript module scripts are the default import type when importing from another JavaScript module; that is, when an `import` statement lacks a `type` import attribute the imported module script's type will be JavaScript. Attempting to import a JavaScript resource using an `import` statement with a `type` import attribute will fail:

```
<script type="module">
    // All of the following will fail, assuming that the imported .mjs files are served with a
    // JavaScript MIME type. JavaScript module scripts are the default and cannot be imported with
    // any import type attribute.
    import foo from "./foo.mjs" with { type: "javascript" };
    import foo2 from "./foo2.mjs" with { type: "js" };
    import foo3 from "./foo3.mjs" with { type: "" };
    await import("./foo4.mjs", { with: { type: null } });
    await import("./foo5.mjs", { with: { type: undefined } });
</script>
```

MDN

8.1.6.7.1 HostGetImportMetaProperties(moduleRecord) §[p1190]

JavaScript contains an implementation-defined HostGetImportMetaProperties abstract operation. User agents must use the following implementation: [JAVASCRIPT][p1583]

1. Let *moduleScript* be *moduleRecord*.[[HostDefined]].
2. Assert: *moduleScript*'s base URL[p1153] is not null, as *moduleScript* is a JavaScript module script[p1153].
3. Let *urlString* be *moduleScript*'s base URL[p1153], serialized.
4. Let *steps* be the following steps, given the argument *specifier*:
    1. Set *specifier* to ? ToString(*specifier*).
    2. Let *url* be the result of resolving a module specifier[p1171] given *moduleScript* and *specifier*.
    3. Return the serialization of *url*.
5. Let *resolveFunction* be ! CreateBuiltinFunction(*steps*, 1, "resolve", « »).
6. Return « Record { [[Key]]: "**url**", [[Value]]: *urlString* }, Record { [[Key]]: "**resolve**", [[Value]]: *resolveFunction* } ».

8.1.6.7.2 HostGetSupportedImportAttributes() §p11 91

JavaScript contains an implementation-defined HostGetSupportedImportAttributes abstract operation. User agents must use the following implementation: [JAVASCRIPT]p1583

1. Return « "type" ».

8.1.6.7.3 HostLoadImportedModule(referrer, moduleRequest, loadState, payload) §p11 91

JavaScript contains an implementation-defined HostLoadImportedModule abstract operation. User agents must use the following implementation: [JAVASCRIPT]p1583

1. Let *settingsObject* be the current settings objectp1151.
2. If *settingsObject*'s global objectp1145 implements WorkletGlobalScopep1342 or ServiceWorkerGlobalScope and *loadState* is undefined:

   Note

   loadState *is undefined when the current fetching process has been initiated by a dynamic* import() *call, either directly or when loading the transitive dependencies of the dynamically imported module.*

   1. Perform FinishLoadingImportedModule(*referrer*, *moduleRequest*, *payload*, ThrowCompletion(a new TypeError)).
   2. Return.
3. Let *referencingScript* be null.
4. Let *originalFetchOptions* be the default script fetch optionsp1154.
5. Let *fetchReferrer* be "client".
6. If *referrer* is a Script Record or a Cyclic Module Record:
   1. Set *referencingScript* to *referrer*.[[HostDefined]].
   2. Set *settingsObject* to *referencingScript*'s settings objectp1152.
   3. Set *fetchReferrer* to *referencingScript*'s base URLp1153.
   4. Set *originalFetchOptions* to *referencingScript*'s fetch optionsp1153.

   Example

   referrer is usually a Script Record or a Cyclic Module Record, but it will not be so for event handlers per the get the current value of the event handlerp1211 algorithm. For example, given:

   ```
   <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, and this operation will receive the current realm as a fallback *referrer*.
7. If *referrer* is a Cyclic Module Record and *moduleRequest* is equal to the first element of *referrer*.[[RequestedModules]]:
   1. For each ModuleRequest record *requested* of *referrer*.[[RequestedModules]]:
      1. If *requested*.[[Attributes]] contains a Record *entry* such that *entry*.[[Key]] is not "type":
         1. Let *error* be a new SyntaxError exception.
         2. If *loadState* is not undefined and *loadState*.[[ErrorToRethrow]] is null, set *loadState*.[[ErrorToRethrow]] to *error*.
         3. Perform FinishLoadingImportedModule(*referrer*, *moduleRequest*, *payload*, ThrowCompletion(*error*)).
         4. Return.

Note

The JavaScript specification re-performs this validation but it is duplicated here to avoid unnecessarily loading any of the dependencies on validation failure.

2. Resolve a module specifier[p1171] given *referencingScript* and *requested*.[[Specifier]], catching any exceptions. If they throw an exception, let *resolutionError* be the thrown exception.
3. If the previous step threw an exception:
    1. If *loadState* is not undefined and *loadState*.[[ErrorToRethrow]] is null, set *loadState*.[[ErrorToRethrow]] to *resolutionError*.
    2. Perform FinishLoadingImportedModule(*referrer*, *moduleRequest*, *payload*, ThrowCompletion(*resolutionError*)).
    3. Return.
4. Let *moduleType* be the result of running the module type from module request[p1164] steps given *requested*.
5. If the result of running the module type allowed[p1164] steps given *moduleType* and *settingsObject* is false:
    1. Let *error* be a new `TypeError` exception.
    2. If *loadState* is not undefined and *loadState*.[[ErrorToRethrow]] is null, set *loadState*.[[ErrorToRethrow]] to *error*.
    3. Perform FinishLoadingImportedModule(*referrer*, *moduleRequest*, *payload*, ThrowCompletion(*error*)).
    4. Return.

Note

This step is essentially validating all of the requested module specifiers and type attributes when the first call to HostLoadImportedModule[p1191] for a static module dependency list is made, to avoid further loading operations in the case any one of the dependencies has a static error. We treat a module with unresolvable module specifiers or unsupported type attributes the same as one that cannot be parsed; in both cases, a syntactic issue makes it impossible to ever contemplate linking the module later.

8. Let *url* be the result of resolving a module specifier[p1171] given *referencingScript* and *moduleRequest*.[[Specifier]], catching any exceptions. If they throw an exception, let *resolutionError* be the thrown exception.
9. If the previous step threw an exception:
    1. If *loadState* is not undefined and *loadState*.[[ErrorToRethrow]] is null, set *loadState*.[[ErrorToRethrow]] to *resolutionError*.
    2. Perform FinishLoadingImportedModule(*referrer*, *moduleRequest*, *payload*, ThrowCompletion(*resolutionError*)).
    3. Return.
10. Let *fetchOptions* be the result of getting the descendant script fetch options[p1155] given *originalFetchOptions*, *url*, and *settingsObject*.
11. Let *destination* be `"script"`.
12. Let *fetchClient* be *settingsObject*.
13. If *loadState* is not undefined:
    1. Set *destination* to *loadState*.[[Destination]].
    2. Set *fetchClient* to *loadState*.[[FetchClient]].
14. Fetch a single imported module script[p1161] given *url*, *fetchClient*, *destination*, *fetchOptions*, *settingsObject*, *fetchReferrer*, *moduleRequest*, and *onSingleFetchComplete* as defined below. If *loadState* is not undefined and *loadState*.[[PerformFetch]] is not null, pass *loadState*.[[PerformFetch]] along as well.

    onSingleFetchComplete given *moduleScript* is the following algorithm:

1. Let *completion* be null.
2. If *moduleScript* is null, then set *completion* to ThrowCompletion(a new TypeError).
3. Otherwise, if *moduleScript*'s parse error[p1153] is not null:
    1. Let *parseError* be *moduleScript*'s parse error[p1153].
    2. Set *completion* to ThrowCompletion(*parseError*).
    3. If *loadState* is not undefined and *loadState*.[[ErrorToRethrow]] is null, set *loadState*.[[ErrorToRethrow]] to *parseError*.
4. Otherwise, set *completion* to NormalCompletion(*moduleScript*'s record[p1152]).
5. Perform FinishLoadingImportedModule(*referrer*, *moduleRequest*, *payload*, *completion*).

8.1.7 Event loops §[p1193]

8.1.7.1 Definitions §[p1193]

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[p1193] of a similar-origin window agent[p1140] is known as a **window event loop**. The event loop[p1193] of a dedicated worker agent[p1140], shared worker agent[p1140], or service worker agent[p1140] is known as a **worker event loop**. And the event loop[p1193] of a worklet agent[p1140] is known as a **worklet event loop**.

Note

Event loops[p1193] do not necessarily correspond to implementation threads. For example, multiple window event loops[p1193] could be cooperatively scheduled in a single thread.

However, for the various worker agents that are allocated with [[CanBlock]] set to true, the JavaScript specification does place requirements on them regarding forward progress, which effectively amount to requiring dedicated per-agent threads in those cases.

An event loop[p1193] has one or more **task queues**. A task queue[p1193] is a set of tasks[p1194].

Note

Task queues[p1193] are sets, not queues, because the event loop processing model[p1196] grabs the first runnable[p1194] task[p1194] from the chosen queue, instead of dequeuing the first task.

Note

The microtask queue[p1194] is not a task queue[p1193].

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.

Note

Not all events are dispatched using the task queue[p1193]; many are dispatched during other tasks.

Parsing

The HTML parser[p1368] 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[p47].

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[p1194], used to group and serialize related tasks.

A *document*

A `Document`[p135] associated with the task, or null for tasks that are not in a window event loop[p1193].

A *script evaluation environment settings object set*

A set of environment settings objects[p1144] used for tracking script evaluation during the task.

A task[p1194] is **runnable** if its document[p1194] is either null or fully active[p1051].

Per its source[p1194] field, each task[p1194] is defined as coming from a specific **task source**. For each event loop[p1193], every task source[p1194] must be associated with a specific task queue[p1193].

Note

Essentially, task sources[p1194] are used within standards to separate logically-different types of tasks, which a user agent might wish to distinguish between. Task queues[p1193] are used by user agents to coalesce task sources within a given event loop[p1193].

Example

For example, a user agent could have one task queue[p1193] for mouse and key events (to which the user interaction task source[p1204] is associated), and another to which all other task sources[p1194] are associated. Then, using the freedom granted in the initial step of the event loop processing model[p1196], 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[p1194] out of order.

Each event loop[p1193] has a **currently running task**, which is either a task[p1194] or null. Initially, this is null. It is used to handle reentrancy.

Each event loop[p1193] has a **microtask queue**, which is a queue of microtasks[p1194], initially empty. A **microtask** is a colloquial way of referring to a task[p1194] that was created via the queue a microtask[p1195] algorithm.

Each event loop[p1193] has a **performing a microtask checkpoint** boolean, which is initially false. It is used to prevent reentrant invocation of the perform a microtask checkpoint[p1200] algorithm.

Each window event loop[p1193] has a `DOMHighResTimeStamp` **last render opportunity time**, initially set to zero.

Each window event loop[p1193] has a `DOMHighResTimeStamp` **last idle period start time**, initially set to zero.

To get the **same-loop windows** for a window event loop[p1193] *loop*, return all `Window`[p965] objects whose relevant agent[p1141]'s event loop[p1193] is *loop*.

8.1.7.2 Queuing tasks §[p1194]

To **queue a task** on a task source[p1194] *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[p1195].

2. If *document* was not given, set *document* to the implied document[p1195].
3. Let *task* be a new task[p1194].
4. Set *task*'s steps[p1194] to *steps*.
5. Set *task*'s source[p1194] to *source*.
6. Set *task*'s document[p1194] to the *document*.
7. Set *task*'s script evaluation environment settings object set[p1194] to an empty set.
8. Let *queue* be the task queue[p1193] to which *source* is associated on *event loop*.
9. Append *task* to *queue*.

⚠Warning!

Failing to pass an event loop and document to queue a task[p1194] means relying on the ambiguous and poorly-specified implied event loop[p1195] and implied document[p1195] concepts. Specification authors should either always pass these values, or use the wrapper algorithms queue a global task[p1195] or queue an element task[p1195] instead. Using the wrapper algorithms is recommended.

To **queue a global task** on a task source[p1194] *source*, with a global object[p1145] *global* and a series of steps *steps*:

1. Let *event loop* be *global*'s relevant agent[p1141]'s event loop[p1193].
2. Let *document* be *global*'s associated `Document`[p966], if *global* is a `Window`[p965] object; otherwise null.
3. Queue a task[p1194] given *source*, *event loop*, *document*, and *steps*.

To **queue an element task** on a task source[p1194] *source*, with an element *element* and a series of steps *steps*:

1. Let *global* be *element*'s relevant global object[p1151].
2. Queue a global task[p1195] given *source*, *global*, and *steps*.

To **queue a microtask** which performs a series of steps *steps*, optionally given a document *document*:

1. Assert: there is a surrounding agent. I.e., this algorithm is not called while in parallel[p44].
2. Let *eventLoop* be the surrounding agent's event loop[p1193].
3. If *document* was not given, set *document* to the implied document[p1195].
4. Let *microtask* be a new task[p1194].
5. Set *microtask*'s steps[p1194] to *steps*.
6. Set *microtask*'s source[p1194] to the **microtask task source**.
7. Set *microtask*'s document[p1194] to *document*.
8. Set *microtask*'s script evaluation environment settings object set[p1194] to an empty set.
9. Enqueue *microtask* on *eventLoop*'s microtask queue[p1194].

Note

It is possible for a microtask[p1194] to be moved to a regular task queue[p1193], if, during its initial execution, it spins the event loop[p1201]. This is the only case in which the source[p1194], document[p1194], and script evaluation environment settings object set[p1194] of the microtask are consulted; they are ignored by the perform a microtask checkpoint[p1200] algorithm.

The **implied event loop** when queuing a task is the one that can 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[p1193]). 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[p1195] concept must not be relied upon and specifications must explicitly provide an event loop[p1193] when queuing a task[p1194].

The **implied document** when queuing a task on an event loop[p1193] *event loop* is determined as follows:

1. If *event loop* is not a window event loop[p1193], 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[p1046], then return the browsing context's active document[p1046].
4. If the task is being queued by or for a script[p1152], then return the script's settings object[p1152]'s global object[p1145]'s associated `Document`[p966].
5. Assert: this step is never reached, because one of the previous conditions is true. Really?

Both implied event loop[p1195] and implied document[p1195] are vaguely-defined and have a lot of action-at-a-distance. The hope is to remove these, especially implied document[p1195]. See issue #4980.

8.1.7.3 Processing model §[p1196]

An event loop[p1193] must continually run through the following steps for as long as it exists:

1. Let *oldestTask* and *taskStartTime* be null.
2. If the event loop[p1193] has a task queue[p1193] with at least one runnable[p1194] task[p1194]:
    1. Let *taskQueue* be one such task queue[p1193], chosen in an implementation-defined manner.

        Note

        Remember that the microtask queue[p1194] is not a task queue[p1193], so it will not be chosen in this step. However, a task queue[p1193] to which the microtask task source[p1195] is associated might be chosen in this step. In that case, the task[p1194] chosen in the next step was originally a microtask[p1194], but it got moved as part of spinning the event loop[p1201].
    2. Set *taskStartTime* to the unsafe shared current time.
    3. Set *oldestTask* to the first runnable[p1194] task[p1194] in *taskQueue*, and remove it from *taskQueue*.
    4. If *oldestTask*'s document[p1194] is not null, then record task start time given *taskStartTime* and *oldestTask*'s document[p1194].
    5. Set the event loop[p1193]'s currently running task[p1194] to *oldestTask*.
    6. Perform *oldestTask*'s steps[p1194].
    7. Set the event loop[p1193]'s currently running task[p1194] back to null.
    8. Perform a microtask checkpoint[p1200].
3. Let *taskEndTime* be the unsafe shared current time. [HRT][p1582]
4. If *oldestTask* is not null:
    1. Let *top-level browsing contexts* be an empty set.
    2. For each environment settings object[p1144] *settings* of *oldestTask*'s script evaluation environment settings object set[p1194]:
        1. Let *global* be *settings*'s global object[p1145].
        2. If *global* is not a `Window`[p965] object, then continue.
        3. If *global*'s browsing context[p966] is null, then continue.
        4. Let *tlbc* be *global*'s browsing context[p966]'s top-level browsing context[p1049].
        5. If *tlbc* is not null, then append it to *top-level browsing contexts*.
    3. Report long tasks, passing in *taskStartTime*, *taskEndTime*, *top-level browsing contexts*, and *oldestTask*.
    4. If *oldestTask*'s document[p1194] is not null, then record task end time given *taskEndTime* and *oldestTask*'s

document[p1194].

5. If this is a window event loop[p1193] that has no *runnable*[p1194] task[p1194] in this event loop[p1193]'s task queues[p1193]:

    1. Set this event loop[p1193]'s last idle period start time[p1194] to the unsafe shared current time.

    2. Let *computeDeadline* be the following steps:

        1. Let *deadline* be this event loop[p1193]'s last idle period start time[p1194] plus 50.

            Note

            The cap of 50ms in the future is to ensure responsiveness to new user input within the threshold of human perception.

        2. Let *hasPendingRenders* be false.

        3. For each *windowInSameLoop* of the same-loop windows[p1194] for this event loop[p1193]:

            1. If *windowInSameLoop*'s map of animation frame callbacks[p1281] is not empty, or if the user agent believes that the *windowInSameLoop* might have pending rendering updates, set *hasPendingRenders* to true.

            2. Let *timerCallbackEstimates* be the result of getting the values of *windowInSameLoop*'s map of active timers[p1256].

            3. For each *timeoutDeadline* of *timerCallbackEstimates*, if *timeoutDeadline* is less than *deadline*, set *deadline* to *timeoutDeadline*.

        4. If *hasPendingRenders* is true:

            1. Let *nextRenderDeadline* be this event loop[p1193]'s last render opportunity time[p1194] plus (1000 divided by the current refresh rate).

                The refresh rate can be hardware- or implementation-specific. For a refresh rate of 60Hz, the *nextRenderDeadline* would be about 16.67ms after the last render opportunity time[p1194].

            2. If *nextRenderDeadline* is less than *deadline*, then return *nextRenderDeadline*.

        5. Return *deadline*.

    3. For each *win* of the same-loop windows[p1194] for this event loop[p1193], perform the start an idle period algorithm for *win* with the following step: return the result of calling *computeDeadline*, coarsened given *win*'s relevant settings object[p1151]'s cross-origin isolated capability[p1144]. [REQUESTIDLECALLBACK][p1585]

6. If this is a worker event loop[p1193]:

    1. If this event loop[p1193]'s agent's single realm's global object[p1145] is a supported[p1281] `DedicatedWorkerGlobalScope`[p1324] and the user agent believes that it would benefit from having its rendering updated at this time:

        1. Let *now* be the current high resolution time given the `DedicatedWorkerGlobalScope`[p1324]. [HRT][p1582]

        2. Run the animation frame callbacks[p1281] for that `DedicatedWorkerGlobalScope`[p1324], 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[p1198] in a window event loop[p1193], a user agent can determine the rate of rendering in the dedicated worker.

    2. If there are no tasks[p1194] in the event loop[p1193]'s task queues[p1193] and the `WorkerGlobalScope`[p1322] object's closing[p1325] flag is true, then destroy the event loop[p1193], aborting these steps, resuming the run a worker[p1328] steps described in the Web workers[p1306] section below.

A window event loop[p1193] *eventLoop* must also run the following in parallel[p44], as long as it exists:

1. Wait until at least one navigable[p1035] whose active document[p1036]'s relevant agent[p1141]'s event loop[p1193] is *eventLoop* might

have a rendering opportunity[p1200].

2. Set *eventLoop*'s last render opportunity time[p1194] to the unsafe shared current time.

3. For each *navigable* that has a rendering opportunity[p1200], queue a global task[p1195] on the rendering task source[p1204] given *navigable*'s active window[p1036] to **update the rendering**:

> **Note**
>
> *This might cause redundant calls to update the rendering[p1198]. However, these calls would have no observable effect because there will be no rendering necessary, as per the* Unnecessary rendering *step. Implementations can introduce further optimizations such as only queuing this task when it is not already queued. However, note that the document associated with the task might become inactive before the task is processed.*

   1. Let *frameTimestamp* be *eventLoop*'s last render opportunity time[p1194].

   2. Let *docs* be all fully active[p1051] `Document`[p135] objects whose relevant agent[p1141]'s event loop[p1193] is *eventLoop*, sorted arbitrarily except that the following conditions must be met:

      - Any `Document`[p135] *B* whose container document[p1039] is *A* must be listed after *A* in the list.
      - If there are two documents *A* and *B* that both have the same non-null container document[p1039] *C*, then the order of *A* and *B* in the list must match the shadow-including tree order of their respective navigable containers[p1038] in *C*'s node tree.

      In the steps below that iterate over *docs*, each `Document`[p135] must be processed in the order it is found in the list.

   3. *Filter non-renderable documents*: Remove from *docs* any `Document`[p135] object *doc* for which any of the following are true:

      - *doc* is render-blocked[p142];
      - *doc*'s visibility state[p864] is "`hidden`";
      - *doc*'s rendering is suppressed for view transitions; or
      - *doc*'s node navigable[p1036] doesn't currently have a rendering opportunity[p1200].

   > **Note**
   >
   > *We have to check for rendering opportunities here, in addition to checking that in the in parallel[p44] steps, as some documents that share the same event loop[p1193] might not have a rendering opportunity[p1200] at the same time.*

   4. *Unnecessary rendering*: Remove from *docs* any `Document`[p135] object *doc* for which all of the following are true:

      - the user agent believes that updating the rendering of *doc*'s node navigable[p1036] would have no visible effect; and
      - *doc*'s map of animation frame callbacks[p1281] is empty.

   5. Remove from *docs* all `Document`[p135] objects for which the user agent believes that it's preferable to skip updating the rendering for other reasons.

   > **Note**
   >
   > *The step labeled* Filter non-renderable documents *prevents the user agent from updating the rendering when it is unable to present new content to the user.*
   >
   > *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[p1194] are executed immediately after each other, with only microtask checkpoints[p1200] interleaved (and without, e.g., animation frame callbacks[p1281] interleaved). Concretely, a user agent might wish to coalesce timer callbacks together, with no intermediate rendering updates.*

   6. For each *doc* of *docs*, reveal[p1103] *doc*.

   7. For each *doc* of *docs*, flush autofocus candidates[p888] for *doc* if its node navigable[p1036] is a top-level traversable[p1037].

8. For each *doc* of *docs*, run the resize steps for *doc*. [CSSOMVIEW]p1581

9. For each *doc* of *docs*, run the scroll steps for *doc*. [CSSOMVIEW]p1581

10. For each *doc* of *docs*, evaluate media queries and report changes for *doc*. [CSSOMVIEW]p1581

11. For each *doc* of *docs*, update animations and send events for *doc*, passing in relative high resolution time given *frameTimestamp* and *doc*'s relevant global objectp1151 as the timestamp. [WEBANIMATIONS]p1587

12. For each *doc* of *docs*, run the fullscreen steps for *doc*. [FULLSCREEN]p1582

13. For each *doc* of *docs*, if the user agent detects that the backing storage associated with a `CanvasRenderingContext2D`p719 or an `OffscreenCanvasRenderingContext2D`p781, *context*, has been lost, then it must run the **context lost steps** for each such *context*:

    1. Let *canvas* be the value of *context*'s `canvas`p724 attribute, if *context* is a `CanvasRenderingContext2D`p719, or the associated `OffscreenCanvas` objectp781 for *context* otherwise.

    2. Set *context*'s context lostp727 to true.

    3. Reset the rendering context to its default statep727 given *context*.

    4. Let *shouldRestore* be the result of firing an event named `contextlost`p1575 at *canvas*, with the `cancelable` attribute initialized to true.

    5. If *shouldRestore* is false, then abort these steps.

    6. Attempt to restore *context* by creating a backing storage using *context*'s attributes and associating them with *context*. If this fails, then abort these steps.

    7. Set *context*'s context lostp727 to false.

    8. Fire an event named `contextrestored`p1575 at *canvas*.

14. For each *doc* of *docs*, run the animation frame callbacksp1281 for *doc*, passing in the relative high resolution time given *frameTimestamp* and *doc*'s relevant global objectp1151 as the timestamp.

15. Let *unsafeStyleAndLayoutStartTime* be the unsafe shared current time.

16. For each *doc* of *docs*:

    1. Let *resizeObserverDepth* be 0.

    2. While true:

        1. Recalculate styles and update layout for *doc*.

        2. Let *hadInitialVisibleContentVisibilityDetermination* be false.

        3. For each element *element* with 'auto' used value of 'content-visibility':

            1. Let *checkForInitialDetermination* be true if *element*'s proximity to the viewport is not determined and it is not relevant to the user. Otherwise, let *checkForInitialDetermination* be false.

            2. Determine proximity to the viewport for *element*.

            3. If *checkForInitialDetermination* is true and *element* is now relevant to the user, then set *hadInitialVisibleContentVisibilityDetermination* to true.

        4. If *hadInitialVisibleContentVisibilityDetermination* is true, then continue.

        Note

        The intent of this step is for the initial viewport proximity determination, which takes effect immediately, to be reflected in the style and layout calculation which is carried out in a previous step of this loop. Proximity determinations other than the initial one take effect at the next rendering opportunityp1200. [CSSCONTAIN]p1580

        5. Gather active resize observations at depth *resizeObserverDepth* for *doc*.

6. If *doc* has active resize observations:

    1. Set *resizeObserverDepth* to the result of broadcasting active resize observations given *doc*.

    2. Continue.

7. Otherwise, break.

3. If *doc* has skipped resize observations, then deliver resize loop error given *doc*.

17. For each *doc* of *docs*, if the focused area[p875] of *doc* is not a focusable area[p874], then run the focusing steps[p881] for *doc*'s viewport, and set *doc*'s relevant global object[p1151]'s navigation API[p995]'s focus changed during ongoing navigation[p1007] to false.

    Example

    For example, this might happen because an element has the `hidden`[p862] attribute added, causing it to stop being rendered[p1488]. It might also happen to an `input`[p539] element when the element gets disabled[p630].

    Note

    This will usually[p882] fire `blur`[p1574] events, and possibly `change`[p1575] events.

    Note

    In addition to this asynchronous fixup, if the focused area of the document[p875] is removed, there is a synchronous fixup[p47]. That one will not *fire `blur`[p1574] or `change`[p1575] events.*

18. For each *doc* of *docs*, perform pending transition operations for *doc*. [CSSVIEWTRANSITIONS][p1582]

19. For each *doc* of *docs*, run the update intersection observations steps for *doc*, passing in the relative high resolution time given *now* and *doc*'s relevant global object[p1151] as the timestamp. [INTERSECTIONOBSERVER][p1583]

20. For each *doc* of *docs*, record rendering time for *doc* given *unsafeStyleAndLayoutStartTime*.

21. For each *doc* of *docs*, mark paint timing for *doc*.

22. For each *doc* of *docs*, update the rendering or user interface of *doc* and its node navigable[p1036] to reflect the current state.

23. For each *doc* of *docs*, process top layer removals given *doc*.

A navigable[p1035] has a **rendering opportunity** if the user agent is currently able to present the contents of the navigable[p1035] 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.

A navigable[p1035]'s rendering opportunities[p1200] are determined based on hardware constraints such as display refresh rates and other factors such as page performance or whether its active document[p1036]'s visibility state[p864] is "`visible`". Rendering opportunities typically occur at regular intervals.

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 navigable[p1035] is not able to sustain this rate, it might drop to a more sustainable 30 rendering opportunities per second for that navigable[p1035], rather than occasionally dropping frames. Similarly, if a navigable[p1035] is not visible, the user agent might decide to drop that page to a much slower 4 rendering opportunities per second, or even less.

When a user agent is to **perform a microtask checkpoint**:

1. If the event loop[p1193]'s performing a microtask checkpoint[p1194] is true, then return.

2. Set the event loop[p1193]'s performing a microtask checkpoint[p1194] to true.

3. While the event loop[p1193]'s microtask queue[p1194] is not empty:

    1. Let *oldestMicrotask* be the result of dequeuing from the event loop[p1193]'s microtask queue[p1194].

2. Set the event loop[p1193]'s currently running task[p1194] to *oldestMicrotask*.
3. Run *oldestMicrotask*.

   Note

   This might involve invoking scripted callbacks, which eventually calls the clean up after running script[p1166] steps, which call this perform a microtask checkpoint[p1200] algorithm again, which is why we use the performing a microtask checkpoint[p1194] flag to avoid reentrancy.

4. Set the event loop[p1193]'s currently running task[p1194] back to null.

4. For each environment settings object[p1144] *settingsObject* whose responsible event loop[p1145] is this event loop[p1193], notify about rejected promises[p1169] given *settingsObject*'s global object[p1145].
5. Cleanup Indexed Database transactions.
6. Perform ClearKeptObjects().

   Note

   When `WeakRef.prototype.deref()` *returns an object, that object is kept alive until the next invocation of ClearKeptObjects(), after which it is again subject to garbage collection.*

7. Set the event loop[p1193]'s performing a microtask checkpoint[p1194] to false.
8. Record timing info for microtask checkpoint.

When an algorithm running in parallel[p44] is to **await a stable state**, the user agent must queue a microtask[p1195] 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. Resume execution of the algorithm in parallel[p44], if appropriate, as described in the algorithm's steps.

Note

Steps in synchronous sections[p1201] 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[p1193]'s currently running task[p1194].

   Note

   task *could be a microtask[p1194].*

2. Let *task source* be *task*'s source[p1194].
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[p1200].

   Note

   If task *is a microtask[p1194] this step will be a no-op due to performing a microtask checkpoint[p1194] being true.*

6. In parallel[p44]:
   1. Wait until the condition *goal* is met.
   2. Queue a task[p1194] 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[p1201] instance in the original algorithm.

Note

This resumes task.

7. Stop *task*, allowing whatever algorithm that invoked it to resume.

Note

This causes the event loop[p1193]'s main set of steps or the perform a microtask checkpoint[p1200] algorithm to continue.

Note

Unlike other algorithms in this and other specifications, which behave similar to programming-language function calls, spin the event loop[p1201] is more like a macro, which saves typing and indentation at the usage site by expanding into a series of steps and operations.

Example

An algorithm whose steps are:

1. Do something.
2. Spin the event loop[p1201] 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[p1200].
5. In parallel[p44]:
    1. Wait until awesomeness happens.
    2. Queue a task[p1194] 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.

Example

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[p1201]:

1. In parallel[p44]:
    1. Do parallel thing 1.
    2. Queue a task[p1194] on the DOM manipulation task source[p1203] to:
        1. Do task thing 1.
        2. Spin the event loop[p1201] until awesomeness happens.
        3. Do task thing 2.
    3. Do parallel thing 2.

The fully expanded version:

1. In parallel[p44]:
    1. Do parallel thing 1.
    2. Let *old stack* be null.
    3. Queue a task[p1194] on the DOM manipulation task source[p1203] 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[p1200].
    4. Wait until awesomeness happens.
    5. Queue a task[p1194] on the DOM manipulation task source[p1203] 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[p1194] until a condition *goal* is met. This means running the following steps:

1. Let *global* be the current global object[p1151].
2. Let *timeBeforePause* be the current high resolution time given *global*.
3. If necessary, update the rendering or user interface of any `Document`[p135] or navigable[p1035] to reflect the current state.
4. Wait until the condition *goal* is met. While a user agent has a paused task[p1194], the corresponding event loop[p1193] must not run further tasks[p1194], and any script in the currently running task[p1194] must block. User agents should remain responsive to user input while paused, however, albeit in a reduced capacity since the event loop[p1193] will not be doing anything.
5. Record pause duration given the duration from *timeBeforePause* to the current high resolution time given *global*.

⚠Warning!

Pausing[p1203] is highly detrimental to the user experience, especially in scenarios where a single event loop[p1193] is shared among multiple documents. User agents are encouraged to experiment with alternatives to pausing[p1203], such as spinning the event loop[p1201] 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[p1193] behavior, including task[p1194] and microtask[p1194] timing. Implementations should continue experimenting even if doing so causes them to violate the exact semantics implied by the pause[p1203] operation.

8.1.7.4 Generic task sources §[p1203]

The following task sources[p1194] are used by a number of mostly unrelated features in this and other specifications.

The *DOM manipulation task source*

This task source[p1194] 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[p47].

The *user interaction task source*

This task source[p1194] 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[p1194] queued[p1194] with the user interaction task source[p1204]. [UIEVENTS][p1587]

The *networking task source*

This task source[p1194] is used for features that trigger in response to network activity.

The *navigation and traversal task source*

This task source[p1194] is used to queue tasks involved in navigation[p1063] and history traversal[p1090].

The *rendering task source*

This task source[p1194] is used solely to update the rendering[p1198].

8.1.7.5 Dealing with the event loop from other specifications §[p1204]

Writing specifications that correctly interact with the event loop[p1193] can be tricky. This is compounded by how this specification uses concurrency-model-independent terminology, so we say things like "event loop[p1193]" and "in parallel[p44]" 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[p1193]. This falls out from the formal event loop processing model[p1196], in that you can eventually trace most algorithms back to a task[p1194] queued[p1194] 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[p692].

From this starting point, the overriding guideline is that any work a specification needs to perform that would otherwise block the event loop[p1193] must instead be performed in parallel[p44] 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[p44], you must not create or manipulate objects associated to a specific realm, global[p1145], or environment settings object[p1144]. (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*[p44] to the JavaScript code.

By extension, you cannot access Web IDL's this value from steps running in parallel[p44], even if those steps were activated by an algorithm that *does* have access to the this value.

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][p1583] [WEBIDL][p1588]

To affect the world of observable JavaScript objects, then, you must queue a global task[p1195] to perform any such manipulations. This ensures your steps are properly interleaved with respect to other things happening on the event loop[p1193]. Furthermore, you must choose a task source[p1194] when queuing a global task[p1195]; this governs the relative order of your steps versus others. If you are unsure which task source[p1194] to use, pick one of the generic task sources[p1203] that sounds most applicable. Finally, you must indicate which global object[p1145] your queued task is associated with; this ensures that if that global object is inactive, the task does not run.

Note

The base primitive, on which queue a global task[p1195] builds, is the queue a task[p1194] algorithm. In general, queue a global task[p1195] is better because it automatically picks the right event loop[p1193] and, where appropriate, document[p1194]. Older specifications often use queue a task[p1194] combined with the implied event loop[p1195] and implied document[p1195] concepts, but this is discouraged.

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[p1193]. This may include converting realm-specific JavaScript values into realm-agnostic specification-level values.
2. Perform a set of potentially-expensive steps in parallel[p44], operating entirely on realm-agnostic values, and producing a realm-agnostic result.
3. Queue a global task[p1195], on a specified task source[p1194] and given an appropriate global object[p1145], to convert the realm-agnostic result back into observable effects on the observable world of JavaScript objects on the event loop[p1193].

¶[p1205] **Example**

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 encoding-parsing a URL[p101] given *string*, relative to the current settings object[p1151].
    2. If *parsed* is failure, then 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.
4. Let *p* be a new promise.
5. Run the following steps in parallel[p44]:
    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 *n*th code unit is equal to *url*'s *n*th code unit plus 13.
        3. Append *encrypted* to *encryptedURLs*.
    3. Queue a global task[p1195] on the networking task source[p1204], given *realm*'s global object[p1145], 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*.
6. Return *p*.

Here are several things to notice about this algorithm:

- It does its URL parsing up front, on the event loop[p1193], before going to the in parallel[p44] steps. This is necessary, since parsing depends on the current settings object[p1151], which would no longer be current after going in parallel[p44].
- Alternately, it could have saved a reference to the current settings object[p1151]'s API base URL[p1144] and used it during the in parallel[p44] 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[p1151], instead of its API base URL[p1144], there would have been a potential race.
- It implicitly passes a list of strings from the initial steps to the in parallel[p44] 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[p44] steps, thus not blocking the main event loop[p1193].

- Promises, as observable JavaScript objects, are never created and manipulated during the in parallel[p44] steps. *p* is created before entering those steps, and then is manipulated during a task[p1194] that is queued[p1195] 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 that is no longer obvious from context.

(On these last two points, see also whatwg/webidl issue #135 and whatwg/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<USVString>`, there was an automatic conversion from realm-specific JavaScript objects provided by the author as input, into the realm-agnostic `sequence<USVString>` 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[p1193] that play a part in this whole process of getting you ready to go in parallel[p44].

8.1.8 Events §[p12 06]

MDN

8.1.8.1 Event handlers §[p12 06]

Many objects can have **event handlers** specified. These act as non-capture event listeners for the object on which they are specified. [DOM][p1582]

An event handler[p1206] is a struct with two items:

- a **value**, which is either null, a callback object, or an internal raw uncompiled handler[p1211]. The `EventHandler`[p1211] callback function type describes how this is exposed to scripts. Initially, an event handler[p1206]'s value[p1206] must be set to null.
- a **listener**, which is either null or an event listener responsible for running the event handler processing algorithm[p1210]. Initially, an event handler[p1206]'s listener[p1206] 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[p1207].

The second way is as an event handler content attribute[p1207]. Event handlers on HTML elements[p46] and some of the event handlers on `Window`[p965] objects are exposed in this way.

For both of these two ways, the event handler[p1206] 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[p1206] is the same as the object on which the corresponding event listener is added. However, the `body`[p212] and `frameset`[p1537] elements expose several event handlers[p1206] that act upon the element's `Window`[p965] object, if one exists. In either case, we call the object an event handler[p1206] acts upon the **target** of that event handler[p1206].

To **determine the target of an event handler**, given an `EventTarget` object *eventTarget* on which the event handler[p1206] is exposed, and an event handler name[p1206] *name*, the following steps are taken:

1. If *eventTarget* is not a `body`[p212] element or a `frameset`[p1537] element, then return *eventTarget*.
2. If *name* is not the name of an attribute member of the `WindowEventHandlers`[p1217] interface mixin and the `Window`-reflecting body element event handler set[p1215] does not contain *name*, then return *eventTarget*.
3. If *eventTarget*'s node document is not an active document[p1046], then return null.

Note

This could happen if this object is a `body`[p212] element without a corresponding `Window`[p965] object, for example.

Note

This check does not necessarily prevent body[p212] and frameset[p1537] elements that are not the body element[p144] of their node document from reaching the next step. In particular, a body[p212] element created in an active document[p1046] (perhaps with document.createElement()) but not connected will also have its corresponding Window[p965] object as the target[p1206] of several event handlers[p1206] exposed through it.

4. Return *eventTarget*'s node document's relevant global object[p1151].

Each EventTarget object that has one or more event handlers[p1206] specified has an associated **event handler map**, which is a map of strings representing names[p1206] of event handlers[p1206] to event handlers[p1206].

When an EventTarget object that has one or more event handlers[p1206] specified is created, its event handler map[p1207] must be initialized such that it contains an entry for each event handler[p1206] that has that object as target[p1206], with items in those event handlers[p1206] set to their initial values.

Note

The order of the entries of event handler map[p1207] could be arbitrary. It is not observable through any algorithms that operate on the map.

Note

Entries are not created in the event handler map[p1207] of an object for event handlers[p1206] that are merely exposed on that object, but have some other object as their targets[p1206].

An **event handler IDL attribute** is an IDL attribute for a specific event handler[p1206]. The name of the IDL attribute is the same as the name[p1206] of the event handler[p1206].

The getter of an event handler IDL attribute[p1207] with name *name*, when called, must run these steps:

1. Let *eventTarget* be the result of determining the target of an event handler[p1206] 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[p1211] given *eventTarget* and *name*.

The setter of an event handler IDL attribute[p1207] with name *name*, when called, must run these steps:

1. Let *eventTarget* be the result of determining the target of an event handler[p1206] given this object and *name*.
2. If *eventTarget* is null, then return.
3. If the given value is null, then deactivate an event handler[p1208] given *eventTarget* and *name*.
4. Otherwise:
    1. Let *handlerMap* be *eventTarget*'s event handler map[p1207].
    2. Let *eventHandler* be *handlerMap*[*name*].
    3. Set *eventHandler*'s value[p1206] to the given value.
    4. Activate an event handler[p1208] given *eventTarget* and *name*.

Note

Certain event handler IDL attributes[p1207] have additional requirements, in particular the onmessage[p1299] attribute of MessagePort[p1299] objects.

An **event handler content attribute** is a content attribute for a specific event handler[p1206]. The name of the content attribute is the same as the name[p1206] of the event handler[p1206].

Event handler content attributes[p1207], 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[p1207] and event handlers[p1206]: [DOM][p1582]

1. If *namespace* is not null, or *localName* is not the name of an event handler content attribute[p1207] on *element*, then return.
2. Let *eventTarget* be the result of determining the target of an event handler[p1206] given *element* and *localName*.
3. If *eventTarget* is null, then return.
4. If *value* is null, then deactivate an event handler[p1208] 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][p1580]
    2. Let *handlerMap* be *eventTarget*'s event handler map[p1207].
    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[p1206] to the internal raw uncompiled handler[p1211] *value*/*location*.
    6. Activate an event handler[p1208] 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][p1582]*

To **deactivate an event handler** given an `EventTarget` object *eventTarget* and a string *name* that is the name[p1206] of an event handler[p1206], run these steps:

1. Let *handlerMap* be *eventTarget*'s event handler map[p1207].
2. Let *eventHandler* be *handlerMap*[*name*].
3. Set *eventHandler*'s value[p1206] to null.
4. Let *listener* be *eventHandler*'s listener[p1206].
5. If *listener* is not null, then remove an event listener with *eventTarget* and *listener*.
6. Set *eventHandler*'s listener[p1206] 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[p1207], then for each *name* → *eventHandler* of *eventTarget*'s associated event handler map[p1207], deactivate an event handler[p1208] given *eventTarget* and *name*.
2. Remove all event listeners given *eventTarget*.

Note

This algorithm is used to define `document.open()`[p1221].

To **activate an event handler** given an `EventTarget` object *eventTarget* and a string *name* that is the name[p1206] of an event handler[p1206], run these steps:

1. Let *handlerMap* be *eventTarget*'s event handler map[p1207].
2. Let *eventHandler* be *handlerMap*[*name*].
3. If *eventHandler*'s listener[p1206] 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[p1210], 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[p1210]. [DOM][p1582]

   Note

   The callback is emphatically not *the event handler[p1206] 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[p1206] to *listener*.

Note

The event listener registration happens only if the event handler[p1206]'s value[p1206] is being set to non-null, and the event handler[p1206] is not already activated. Since listeners are called in the order they were registered, assuming no deactivation[p1208] 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[p1206]'s value[p1206] 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[p1206]'s value[p1206] 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.

```
<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'); };
 button.addEventListener('click', function () { alert('FOUR') }, false);
</script>
```

However, in the following example, the event handler is deactivated[p1208] 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.

```
<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[p1206] is triggered or not.

The event handler processing algorithm for an `EventTarget` object *eventTarget*, a string *name* representing the name[p1206] of an event handler[p1206], and an `Event` object *event* is as follows:

1. If the result of WebDriver BiDi scripting is enabled with *eventTarget*'s relevant settings object[p1151] is false, then return.
2. Let *callback* be the result of getting the current value of the event handler[p1211] given *eventTarget* and *name*.
3. If *callback* is null, then return.
4. If *callback*'s associated realm's global object[p1145] is a `Window`[p965] object whose associated `Document`[p966] is not fully active[p1051], then return.
5. Let *special error event handling* be true if *event* is an `ErrorEvent`[p1168] object, *event*'s `type` is "`error`[p1575]", and *event*'s `currentTarget` implements the `WindowOrWorkerGlobalScope`[p1218] mixin. Otherwise, let *special error event handling* be false.
6. Process the `Event` object *event* as follows:

   ↪ **If *special error event handling* is true**

   Let *return value* be the result of invoking *callback* with « *event*'s `message`[p1169], *event*'s `filename`[p1169], *event*'s `lineno`[p1169], *event*'s `colno`[p1169], *event*'s `error`[p1169] », "`rethrow`", and with *callback this value* set to *event*'s `currentTarget`.

   ↪ **Otherwise**

   Let *return value* be the result of invoking *callback* with « *event* », "`rethrow`", and with *callback this value* set to *event*'s `currentTarget`.

   Note

   If an exception gets thrown by the callback, it will be rethrown, ending these steps. The exception will propagate to the DOM event dispatch logic, which will then report[p1167] it.

7. Process *return value* as follows:

   ↪ **If *event* is a `BeforeUnloadEvent`[p1030] object and *event*'s `type` is "`beforeunload`[p1574]"**

   Note

   In this case, the event handler IDL attribute[p1207]'s type will be `OnBeforeUnloadEventHandler`[p1211], so return value *will have been coerced into either null or a `DOMString`.*

   If *return value* is not null:

   1. Set *event*'s canceled flag.
   2. If *event*'s `returnValue`[p1030] attribute's value is the empty string, then set *event*'s `returnValue`[p1030] 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`[p1574]" *but* event *is* not *a*

BeforeUnloadEvent[p1030] *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[p1211] 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:

```
<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[p1215] *handlers on global objects, where returning* true *cancels the event.*
- *The* onbeforeunload[p1215] *handler, where returning any non-null and non-undefined value will cancel the event.*

For historical reasons, the onerror[p1215] 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[p1215] 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[p1206] of an event handler[p1206], it must run these steps:

1. Let *handlerMap* be *eventTarget*'s event handler map[p1207].
2. Let *eventHandler* be *handlerMap*[*name*].
3. If *eventHandler*'s value[p1206] is an internal raw uncompiled handler[p1211]:
    1. If *eventTarget* is an element, then let *element* be *eventTarget*, and *document* be *element*'s node document. Otherwise, *eventTarget* is a `Window`[p965] object, let *element* be null, and *document* be *eventTarget*'s associated `Document`[p966].
    2. If *document*'s browsing context[p1046] is null, or *document*'s active sandboxing flag set[p959] has its sandboxed scripts browsing context flag[p957] set, then return null.
    3. Let *body* be the uncompiled script body in *eventHandler*'s value[p1206].
    4. Let *location* be the location where the script body originated, as given by *eventHandler*'s value[p1206].
    5. If *element* is not null and *element* has a form owner[p626], let *form owner* be that form owner[p626]. Otherwise, let *form owner* be null.
    6. Let *settings object* be the relevant settings object[p1151] 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[p1206] to null.

            Note

            This does not deactivate[p1208] the event handler, which additionally removes the event handler's listener[p1206] (if present).
        2. Let *syntaxError* be a new `SyntaxError` exception associated with *settings object*'s realm[p1145] which describes the error while parsing. It should be based on *location*, where the script body originated.
        3. Report an exception[p1167] with *syntaxError* for *settings object*'s global object[p1145].
        4. Return null.
    8. Push *settings object*'s realm execution context[p1144] 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 realm.
    9. Let *function* be the result of calling OrdinaryFunctionCreate, with arguments:

        functionPrototype

        %Function.prototype%

        sourceText

        ↪ **If *name* is `onerror`[p1215] and *eventTarget* is a `Window`[p965] 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`[p1215] and *eventTarget* is a `Window`[p965] 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[p1145].
2. Let *scope* be *realm*.[[GlobalEnv]].
3. If *eventHandler* is an element's event handler[p1206], then set *scope* to NewObjectEnvironment(*document*, true, *scope*).

   (Otherwise, *eventHandler* is a `Window`[p965] object's event handler[p1206].)
4. If *form owner* is not null, then set *scope* to NewObjectEnvironment(*form owner*, true, *scope*).
5. If *element* is not null, then set *scope* to NewObjectEnvironment(*element*, true, *scope*).
6. Return *scope*.

10. Remove *settings object*'s realm execution context[p1144] from the JavaScript execution context stack.
11. Set *function*.[[ScriptOrModule]] to null.

    Note

    This is done because the default behavior, of associating the created function with the nearest script[p1152] 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 [[ScriptOrModule]] (since then this algorithm would be first invoked when the active script[p1154] is null), whereas one that is first invoked by dispatching an event from script would have its [[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[p1153] of the associated script. Nulling out [[ScriptOrModule]] means that HostLoadImportedModule[p1191] will fall back to the current settings object[p1151]'s API base URL[p1144].
12. Set *eventHandler*'s value[p1206] to the result of creating a Web IDL `EventHandler`[p1211] callback function object whose object reference is *function* and whose callback context is *settings object*.

4. Return *eventHandler*'s value[p1206].

8.1.8.2 Event handlers on elements, `Document`[p135] objects, and `Window`[p965] objects §[p12 13]

The following are the event handlers[p1206] (and their corresponding event handler event types[p1209]) that must be supported by all HTML elements[p46], as both event handler content attributes[p1207] and event handler IDL attributes[p1207]; and that must be supported by all `Document`[p135] and `Window`[p965] objects, as event handler IDL attributes[p1207]:

Event handler[p1206]	Event handler event type[p1209]
onabort	abort
onauxclick	auxclick
onbeforeinput	beforeinput
onbeforematch	beforematch[p1574]
onbeforetoggle	beforetoggle[p1574]
oncancel	cancel[p1575]
oncanplay	canplay[p485]
oncanplaythrough	canplaythrough[p485]
onchange	change[p1575]
onclick	click
onclose	close[p1575]
oncommand	command[p1575]
oncontextlost	contextlost[p1575]

Event handler[p1206]	Event handler event type[p1209]
oncontextmenu	contextmenu
oncontextrestored	contextrestored[p1575]
oncopy	copy
oncuechange	cuechange[p486]
oncut	cut
ondblclick	dblclick
ondrag	drag[p923]
ondragend	dragend[p924]
ondragenter	dragenter[p924]
ondragleave	dragleave[p924]
ondragover	dragover[p924]
ondragstart	dragstart[p923]
ondrop	drop[p924]
ondurationchange	durationchange[p485]
onemptied	emptied[p485]
onended	ended[p485]
onformdata	formdata[p1575]
oninput	input
oninvalid	invalid[p1575]
onkeydown	keydown
onkeypress	keypress
onkeyup	keyup
onloadeddata	loadeddata[p485]
onloadedmetadata	loadedmetadata[p485]
onloadstart	loadstart[p484]
onmousedown	mousedown
onmouseenter	mouseenter
onmouseleave	mouseleave
onmousemove	mousemove
onmouseout	mouseout
onmouseover	mouseover
onmouseup	mouseup
onpaste	paste
onpause	pause[p485]
onplay	play[p485]
onplaying	playing[p485]
onprogress	progress[p484]
onratechange	ratechange[p485]
onreset	reset[p1576]
onscrollend	scrollend
onsecuritypolicyviolation	securitypolicyviolation
onseeked	seeked[p485]
onseeking	seeking[p485]
onselect	select[p1576]
onslotchange	slotchange
onstalled	stalled[p485]
onsubmit	submit[p1576]
onsuspend	suspend[p484]
ontimeupdate	timeupdate[p485]
ontoggle	toggle[p1576]
onvolumechange	volumechange[p486]
onwaiting	waiting[p485]
onwebkitanimationend	webkitAnimationEnd
onwebkitanimationiteration	webkitAnimationIteration
onwebkitanimationstart	webkitAnimationStart

Event handler[p1206]	Event handler event type[p1209]
onwebkittransitionend	webkitTransitionEnd
onwheel	wheel

The following are the event handlers[p1206] (and their corresponding event handler event types[p1209]) that must be supported by all HTML elements[p46] other than body[p212] and frameset[p1537] elements, as both event handler content attributes[p1207] and event handler IDL attributes[p1207]; that must be supported by all Document[p135] objects, as event handler IDL attributes[p1207]; and that must be supported by all Window[p965] objects, as event handler IDL attributes[p1207] on the Window[p965] objects themselves, and with corresponding event handler content attributes[p1207] and event handler IDL attributes[p1207] exposed on all body[p212] and frameset[p1537] elements that are owned by that Window[p965] object's associated Document[p966]:

Event handler[p1206]	Event handler event type[p1209]
onblur	blur[p1574]
onerror	error[p1575]
onfocus	focus[p1575]
onload	load[p1575]
onresize	resize
onscroll	scroll

We call the set of the names[p1206] of the event handlers[p1206] listed in the first column of this table the **Window-reflecting body element event handler set**.

The following are the event handlers[p1206] (and their corresponding event handler event types[p1209]) that must be supported by Window[p965] objects, as event handler IDL attributes[p1207] on the Window[p965] objects themselves, and with corresponding event handler content attributes[p1207] and event handler IDL attributes[p1207] exposed on all body[p212] and frameset[p1537] elements that are owned by that Window[p965] object's associated Document[p966]:

Event handler[p1206]	Event handler event type[p1209]
onafterprint	afterprint[p1574]
onbeforeprint	beforeprint[p1574]
onbeforeunload	beforeunload[p1574]
onhashchange	hashchange[p1575]
onlanguagechange	languagechange[p1575]
onmessage	message[p1575]
onmessageerror	messageerror[p1575]
onoffline	offline[p1576]
ononline	online[p1576]
onpageswap	pageswap[p1576]
onpagehide	pagehide[p1576]
onpagereveal	pagereveal[p1576]
onpageshow	pageshow[p1576]
onpopstate	popstate[p1576]
onrejectionhandled	rejectionhandled[p1576]
onstorage	storage[p1576]
onunhandledrejection	unhandledrejection[p1576]
onunload	unload[p1576]

This list of event handlers[p1206] is reified as event handler IDL attributes[p1207] through the WindowEventHandlers[p1217] interface mixin.

The following are the event handlers[p1206] (and their corresponding event handler event types[p1209]) that must be supported on Document[p135] objects as event handler IDL attributes[p1207]:

Event handler[p1206]	Event handler event type[p1209]
onreadystatechange	readystatechange[p1576]
onvisibilitychange	visibilitychange[p1576]

8.1.8.2.1 IDL definitions §[p12 16]

IDL

```
interface mixin GlobalEventHandlers {
  attribute EventHandler onabort;
  attribute EventHandler onauxclick;
  attribute EventHandler onbeforeinput;
  attribute EventHandler onbeforematch;
  attribute EventHandler onbeforetoggle;
  attribute EventHandler onblur;
  attribute EventHandler oncancel;
  attribute EventHandler oncanplay;
  attribute EventHandler oncanplaythrough;
  attribute EventHandler onchange;
  attribute EventHandler onclick;
  attribute EventHandler onclose;
  attribute EventHandler oncommand;
  attribute EventHandler oncontextlost;
  attribute EventHandler oncontextmenu;
  attribute EventHandler oncontextrestored;
  attribute EventHandler oncopy;
  attribute EventHandler oncuechange;
  attribute EventHandler oncut;
  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 onmousedown;
  [LegacyLenientThis] attribute EventHandler onmouseenter;
  [LegacyLenientThis] attribute EventHandler onmouseleave;
  attribute EventHandler onmousemove;
  attribute EventHandler onmouseout;
  attribute EventHandler onmouseover;
  attribute EventHandler onmouseup;
  attribute EventHandler onpaste;
  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 onscrollend;
  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 onpagereveal;
  attribute EventHandler onpageshow;
  attribute EventHandler onpageswap;
  attribute EventHandler onpopstate;
  attribute EventHandler onrejectionhandled;
  attribute EventHandler onstorage;
  attribute EventHandler onunhandledrejection;
  attribute EventHandler onunload;
};
```

8.1.8.3 Event firing §p12 17

Certain operations and methods are defined as firing events on elements. For example, the `click()` p871 method on the `HTMLElement` p149 interface is defined as firing a `click` event on the element. [POINTEREVENTS] p1585

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` p965 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`[p1217] at *target*.

8.2 The `WindowOrWorkerGlobalScope`[p1218] mixin §[p12 18]

The `WindowOrWorkerGlobalScope`[p1218] mixin is for use of APIs that are to be exposed on `Window`[p965] and `WorkerGlobalScope`[p1322] objects.

Note

Other standards are encouraged to further extend it using `partial interface mixin WindowOrWorkerGlobalScope { … };` *along with an appropriate reference.*

IDL

```
typedef (DOMString or Function or TrustedScript) TimerHandler;

interface mixin WindowOrWorkerGlobalScope {
  [Replaceable] readonly attribute USVString origin;
  readonly attribute boolean isSecureContext;
  readonly attribute boolean crossOriginIsolated;

  undefined reportError(any e);

  // base64 utility methods
  DOMString btoa(DOMString data);
  ByteString atob(DOMString data);

  // timers
  long setTimeout(TimerHandler handler, optional long timeout = 0, any... arguments);
  undefined clearTimeout(optional long id = 0);
  long setInterval(TimerHandler handler, optional long timeout = 0, any... arguments);
  undefined clearInterval(optional long id = 0);

  // microtask queuing
  undefined queueMicrotask(VoidFunction callback);

  // ImageBitmap
  Promise<ImageBitmap> createImageBitmap(ImageBitmapSource image, optional ImageBitmapOptions options =
{});
  Promise<ImageBitmap> createImageBitmap(ImageBitmapSource image, long sx, long sy, long sw, long sh,
optional ImageBitmapOptions options = {});

  // structured cloning
  any structuredClone(any value, optional StructuredSerializeOptions options = {});
};
Window includes WindowOrWorkerGlobalScope;
WorkerGlobalScope includes WindowOrWorkerGlobalScope;
```

For web developers (non-normative)

`self.isSecureContext`[p1219]

Returns whether or not this global object represents a secure context[p1152]. [SECURE-CONTEXTS][p1586]

`self.origin`[p1219]

Returns the global object's origin[p939], serialized as string.

`self.crossOriginIsolated`[p1219]

Returns whether scripts running in this global are allowed to use APIs that require cross-origin isolation. This depends on the

`Cross-Origin-Opener-Policy[p946]` and `Cross-Origin-Embedder-Policy[p955]` HTTP response headers and the "cross-origin-isolated[p79]" feature.

Example

Developers are strongly encouraged to use `self.origin` over `location.origin`. The former returns the origin[p939] of the environment, the latter of the URL of the environment. Imagine the following script executing in a document on `https://stargate.example/`:

```
var frame = document.createElement("iframe")
frame.onload = function() {
  var frameWin = frame.contentWindow
  console.log(frameWin.location.origin) // "null"
  console.log(frameWin.origin) // "https://stargate.example"
}
document.body.appendChild(frame)
```

`self.origin` is a more reliable security indicator.

The **isSecureContext** getter steps are to return true if this's relevant settings object[p1151] is a secure context[p1152], or false otherwise.

The **origin** getter steps are to return this's relevant settings object[p1151]'s origin[p1144], serialized[p939].

The **crossOriginIsolated** getter steps are to return this's relevant settings object[p1151]'s cross-origin isolated capability[p1144].

An element implementing the WindowOrWorkerGlobalScope[p1218] mixin has the following extract an origin[p943] steps:

1. If this's relevant settings object[p1151]'s origin[p1144] is not same origin-domain[p940] with the entry settings object[p1148]'s origin[p1144], then return null.
2. Return this's relevant settings object[p1151]'s origin[p1144].

Note

Since these objects are potentially accessible cross-origin (e.g., through WindowProxy[p977]), we need a security check here before granting access to the origin.

8.3 Base64 utility methods §[p1219]

The atob()[p1220] and btoa()[p1220] methods allow developers to transform content to and from the base64 encoding.

Note

In these APIs, for mnemonic purposes, the "b" can be considered to stand for "binary", and the "a" for "ASCII". In practice, though, for primarily historical reasons, both the input and output of these functions are Unicode strings.

For web developers (non-normative)

***result* = self.btoa[p1220](*data*)**

Takes the input data, in the form of a Unicode string containing only characters in the range U+0000 to U+00FF, each representing a binary byte with values 0x00 to 0xFF respectively, and converts it to its base64 representation, which it returns.

Throws an "InvalidCharacterError" DOMException if the input string contains any out-of-range characters.

***result* = self.atob[p1220](*data*)**

Takes the input data, in the form of a Unicode string containing base64-encoded binary data, decodes it, and returns a string consisting of characters in the range U+0000 to U+00FF, each representing a binary byte with values 0x00 to 0xFF respectively, corresponding to that binary data.

Throws an "InvalidCharacterError" DOMException if the input string is not valid base64 data.

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 steps are:

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 §p1220

Note

APIs for dynamically inserting markup into the document interact with the parser, and thus their behavior varies depending on whether they are used with HTML documents (and the HTML parserp1368) or XML documents (and the XML parserp1484).

Documentp135 objects have a **throw-on-dynamic-markup-insertion counter**, which is used in conjunction with the create an element for the tokenp1418 algorithm to prevent custom element constructorsp796 from being able to use document.open()p1221, document.close()p1222, and document.write()p1223 when they are invoked by the parser. Initially, the counter must be set to zero.

8.4.1 Opening the input stream §p1220

For web developers (non-normative)

***document* = *document*.open()p1221**

Causes the Documentp135 to be replaced in-place, as if it was a new Documentp135 object, but reusing the previous object, which is then returned.

The resulting Documentp135 has an HTML parser associated with it, which can be given data to parse using document.write()p1223.

The method has no effect if the Documentp135 is still being parsed.

Throws an "InvalidStateError" DOMException if the Documentp135 is an XML document.

Throws an "InvalidStateError" DOMException if the parser is currently executing a custom element constructorp796.

***window* = *document*.open(*url*, *name*, *features*)p1222**

Works like the window.open()p969 method.

Documentp135 objects have an **active parser was aborted** boolean, which is used to prevent scripts from invoking the document.open()p1221 and document.write()p1223 methods (directly or indirectly) after the document's active parserp141 has been aborted. It is initially false.

The **document open steps**, given a *document*, are as follows:

1. If *document* is an XML document, then throw an "InvalidStateError" DOMException.
2. If *document*'s throw-on-dynamic-markup-insertion counterp1220 is greater than 0, then throw an "InvalidStateError" DOMException.
3. Let *entryDocument* be the entry global objectp1148's associated Documentp966.
4. If *document*'s origin is not same originp940 to *entryDocument*'s origin, then throw a "SecurityError" DOMException.
5. If *document* has an active parserp141 whose script nesting levelp1370 is greater than 0, then return *document*.

Note

This basically causes document.open()p1221 to be ignored when it's called in an inline script found during parsing, while

still letting it have an effect when called from a non-parser task such as a timer callback or event handler.

6. Similarly, if *document*'s unload counter[p1114] is greater than 0, then return *document*.

Note

This basically causes `document.open()`[p1221] *to be ignored when it's called from a* `beforeunload`[p1574], `pagehide`[p1576], *or* `unload`[p1576] *event handler while the* `Document`[p135] *is being unloaded.*

7. If *document*'s active parser was aborted[p1220] is true, then return *document*.

Note

This notably causes `document.open()`[p1221] *to be ignored if it is called after a navigation*[p1063] *has started, but only during the initial parse. See issue #4723 for more background.*

8. If *document*'s node navigable[p1036] is non-null and *document*'s node navigable[p1036]'s ongoing navigation[p1076] is a navigation ID[p1062], then stop loading[p1118] *document*'s node navigable[p1036].

9. For each shadow-including inclusive descendant *node* of *document*, erase all event listeners and handlers[p1208] given *node*.

10. If *document* is the associated `Document`[p966] of *document*'s relevant global object[p1151], then erase all event listeners and handlers[p1208] given *document*'s relevant global object[p1151].

11. Replace all with null within *document*.

12. If *document* is fully active[p1051]:

    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[p1078] with *document* and *newURL*.

13. Set *document*'s is initial `about:blank`[p136] to false.

14. If *document*'s iframe load in progress[p408] flag is set, then set *document*'s mute iframe load[p408] flag.

15. Set *document* to no-quirks mode.

16. Create an HTML parser[p1368] whose allow declarative shadow roots[p1387] is *document*'s allow declarative shadow roots, and associate it with *document*. This is a **script-created parser** (meaning that it can be closed by the `document.open()`[p1221] and `document.close()`[p1222] methods, and that the tokenizer will wait for an explicit call to `document.close()`[p1222] before emitting an end-of-file token). The encoding confidence[p1375] is *irrelevant*.

17. Set the insertion point[p1382] to point at just before the end of the input stream[p1382] (which at this point will be empty).

18. Update the current document readiness[p141] of *document* to "`loading`".

Note

This causes a `readystatechange`[p1576] *event to fire, but the event is actually unobservable to author code, because of the previous step which erased all event listeners and handlers*[p1208] *that could observe it.*

19. Return *document*.

Note

The document open steps[p1220] *do not affect whether a* `Document`[p135] *is ready for post-load tasks*[p1459] *or completely loaded*[p1114].

The **`open(unused1, unused2)`** method must return the result of running the document open steps[p1220] with this.

¶[p1221] 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. whatwg/webidl issue #581 investigates changing the algorithm to allow for their removal. [WEBIDL]*[p1588]

The **`open(url, name, features)`** method must run these steps:

1. If this is not fully active[p1051], then throw an "InvalidAccessError" DOMException.
2. Return the result of running the window open steps[p967] with *url*, *name*, and *features*.

8.4.2 Closing the input stream §[p1222]

For web developers (non-normative)

***document*.close[p1222]()**

Closes the input stream that was opened by the document.open()[p1221] method.

Throws an "InvalidStateError" DOMException if the Document[p135] is an XML document.

Throws an "InvalidStateError" DOMException if the parser is currently executing a custom element constructor[p796].

The **`close()`** method must run the following steps:

1. If this is an XML document, then throw an "InvalidStateError" DOMException.
2. If this's throw-on-dynamic-markup-insertion counter[p1220] is greater than zero, then throw an "InvalidStateError" DOMException.
3. If there is no script-created parser[p1221] associated with this, then return.
4. Insert an explicit "EOF" character[p1382] at the end of the parser's input stream[p1382].
5. If this's pending parsing-blocking script[p698] is not null, then return.
6. Run the tokenizer, processing resulting tokens as they are emitted, and stopping when the tokenizer reaches the explicit "EOF" character[p1382] or spins the event loop[p1201].

8.4.3 document.write()[p1223] §[p1222]

For web developers (non-normative)

***document*.write[p1223](...*text*)**

In general, adds the given string(s) to the Document[p135]'s input stream.

⚠Warning!

This method has very idiosyncratic behavior. In some cases, this method can affect the state of the HTML parser[p1368] while the parser is running, resulting in a DOM that does not correspond to the source of the document (e.g. if the string written is the string "<plaintext>" or "<!--"). In other cases, the call can clear the current page first, as if document.open()[p1221] had been called. In yet more cases, the method is simply ignored, or throws an exception. User agents are explicitly allowed to avoid executing script elements inserted via this method[p1428]. 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[p796].

⚠Warning!

This method performs no sanitization to remove potentially-dangerous elements and attributes like script[p685] or event handler content attributes[p1207].

Document[p135] objects have an **ignore-destructive-writes counter**, which is used in conjunction with the processing of script[p685] elements to prevent external scripts from being able to use document.write()[p1223] to blow away the document by implicitly calling document.open()[p1221]. Initially, the counter must be set to zero.

The **document write steps**, given a Document[p135] object *document*, a list *text*, a boolean *lineFeed*, and a string *sink*, are as follows:

1. Let *string* be the empty string.
2. Let *isTrusted* be false if *text* contains a string; otherwise true.
3. For each *value* of *text*:
    1. If *value* is a TrustedHTML object, then append *value*'s associated data to *string*.
    2. Otherwise, append *value* to *string*.
4. If *isTrusted* is false, set *string* to the result of invoking the get trusted type compliant string algorithm with TrustedHTML, this's relevant global object[p1151], *string*, *sink*, and "script".
5. If *lineFeed* is true, append U+000A LINE FEED to *string*.
6. If *document* is an XML document, then throw an "InvalidStateError" DOMException.
7. If *document*'s throw-on-dynamic-markup-insertion counter[p1220] is greater than 0, then throw an "InvalidStateError" DOMException.
8. If *document*'s active parser was aborted[p1220] is true, then return.
9. If the insertion point[p1382] is undefined:
    1. If *document*'s unload counter[p1114] is greater than 0 or *document*'s ignore-destructive-writes counter[p1222] is greater than 0, then return.
    2. Run the document open steps[p1220] with *document*.
10. Insert *string* into the input stream[p1382] just before the insertion point[p1382].
11. If *document*'s pending parsing-blocking script[p698] is null, then have the HTML parser[p1368] process *string*, 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[p685] end tag token is emitted by the tokenizer).

    Note

    If the document.write()[p1223] method was called from script executing inline (i.e. executing because the parser parsed a set of script[p685] tags), then this is a reentrant invocation of the parser[p1369]. If the parser pause flag[p1370] is set, the tokenizer will abort immediately and no HTML will be parsed, per the tokenizer's parser pause flag check[p1388].

The **document.write(...*text*)** method steps are to run the document write steps[p1223] with this, *text*, false, and "Document write".

8.4.4 document.writeln()[p1224] §[p12 23]

For web developers (non-normative)

***document*.writeln[p1224](...*text*)**

Adds the given string(s) to the Document[p135]'s input stream, followed by a newline character. If necessary, calls the open()[p1221] method implicitly first.

⚠Warning!

This method has very idiosyncratic behavior. Use of this method is strongly discouraged, for the same reasons as document.write()[p1223].

Throws an "InvalidStateError" DOMException when invoked on XML documents.

Throws an "InvalidStateError" DOMException if the parser is currently executing a custom element constructor[p796].

⚠Warning!

This method performs no sanitization to remove potentially-dangerous elements and attributes like script[p685] or event handler content attributes[p1207].

The **document.writeln(...*text*)** method steps are to run the document write steps[p1223] with this, *text*, true, and "Document writeln".

8.5 DOM parsing and serialization APIs §[p1224]

IDL

```
partial interface Element {
  [CEReactions] undefined setHTML(DOMString html, optional SetHTMLOptions options = {});
  [CEReactions] undefined setHTMLUnsafe((TrustedHTML or DOMString) html, optional SetHTMLUnsafeOptions
options = {});
  DOMString getHTML(optional GetHTMLOptions options = {});

  [CEReactions] attribute (TrustedHTML or [LegacyNullToEmptyString] DOMString) innerHTML;
  [CEReactions] attribute (TrustedHTML or [LegacyNullToEmptyString] DOMString) outerHTML;
  [CEReactions] undefined insertAdjacentHTML(DOMString position, (TrustedHTML or DOMString) string);
};

partial interface ShadowRoot {
  [CEReactions] undefined setHTML(DOMString html, optional SetHTMLOptions options = {});
  [CEReactions] undefined setHTMLUnsafe((TrustedHTML or DOMString) html, optional SetHTMLUnsafeOptions
options = {});
  DOMString getHTML(optional GetHTMLOptions options = {});

  [CEReactions] attribute (TrustedHTML or [LegacyNullToEmptyString] DOMString) innerHTML;
};

enum SanitizerPresets { "default" };
dictionary SetHTMLOptions {
  (Sanitizer or SanitizerConfig or SanitizerPresets) sanitizer = "default";
};
dictionary SetHTMLUnsafeOptions {
  (Sanitizer or SanitizerConfig or SanitizerPresets) sanitizer = {};
  boolean runScripts = false;
};
dictionary ParseHTMLUnsafeOptions {
  (Sanitizer or SanitizerConfig or SanitizerPresets) sanitizer = {};
};

dictionary GetHTMLOptions {
  boolean serializableShadowRoots = false;
  sequence<ShadowRoot> shadowRoots = [];
};
```

8.5.1 The DOMParser[p1225] interface §[p1224]

The DOMParser[p1225] interface allows authors to create new Document[p135] objects by parsing strings, as either HTML or XML.

For web developers (non-normative)

***parser* = new DOMParser[p1225]()**

Constructs a new DOMParser[p1225] object.

***document* = *parser*.parseFromString[p1225](*string*, *type*)**

Parses *string* using either the HTML or XML parser, according to *type*, and returns the resulting Document[p135]. *type* can be "text/html[p1546]" (which will invoke the HTML parser), or any of "text/xml[p1578]", "application/xml[p1577]", "application/xhtml+xml[p1548]", or "image/svg+xml[p1577]" (which will invoke the XML parser).

For the XML parser, if *string* cannot be parsed, then the returned Document[p135] will contain elements describing the resulting

error.

Note that `script`[p685] elements are not evaluated during parsing, and the resulting document's encoding will always be UTF-8. The document's URL will be inherited from *parser*'s relevant global object[p1151].

Values other than the above for *type* will cause a `TypeError` exception to be thrown.

Note

The design of `DOMParser`[p1225], as a class that needs to be constructed and then have its `parseFromString()`[p1225] method called, is an unfortunate historical artifact. If we were designing this functionality today it would be a standalone function. For parsing HTML, the modern alternative is `Document.parseHTMLUnsafe()`[p1227].

⚠Warning!

This method performs no sanitization to remove potentially-dangerous elements and attributes like `script`[p685] or event handler content attributes[p1207].

IDL

```
[Exposed=Window]
interface DOMParser {
  constructor();

  [NewObject] Document parseFromString((TrustedHTML or DOMString) string, DOMParserSupportedType type);
};

enum DOMParserSupportedType {
  "text/html",
  "text/xml",
  "application/xml",
  "application/xhtml+xml",
  "image/svg+xml"
};
```

The **`new DOMParser()`** constructor steps are to do nothing.

The **`parseFromString(string, type)`** method steps are:

1. Let *compliantString* be the result of invoking the get trusted type compliant string algorithm with `TrustedHTML`, this's relevant global object[p1151], *string*, "`DOMParser parseFromString`", and "`script`".
2. Let *relevantDocument* be this's relevant global object[p1151]'s associated `Document`[p966].
3. Let *document* be a new `Document`[p135], whose content type is *type*, origin is *relevantDocument*'s origin, and URL is *relevantDocument*'s URL.

   Note

   The document's encoding will be left as its default, of UTF-8. In particular, any XML declarations or `meta`[p196] elements found while parsing compliantString *will have no effect.*

4. Switch on *type*:

   ↪ **"`text/html`"**

   1. Parse HTML from a string[p1226] given *document* and *compliantString*.

      Note

      Since document *does not have a browsing context[p1046], scripting is disabled[p1152].*

   ↪ ***Otherwise***

   1. Create an XML parser[p1484] *parser*, associated with *document*, and with XML scripting support disabled[p1485].
   2. Parse *compliantString* using *parser*.

3. If the previous step resulted in an XML well-formedness or XML namespace well-formedness error:

    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*.

5. Return *document*.

To **parse HTML from a string**, given a Document[p135] *document* and a string *html*:

1. Set *document*'s type to "`html`".

2. Let *parser* be a new HTML parser[p1368] whose allow declarative shadow roots[p1387] is *document*'s allow declarative shadow roots, associated with *document*.

3. Place *html* into the input stream[p1382] for *parser*. The encoding confidence[p1375] 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.

8.5.2 HTML parsing methods §[p1226]

For web developers (non-normative)

***element*.setHTML[p1227](*html*, *options*)**

Parses *html* using the HTML parser with options *options*, and replaces the children of *element* with the result. *element* provides context for the HTML parser. The parsed fragment is sanitized[p1243] based on the *options*'s "sanitizer[p1224]" member, and unsafe content is removed[p1240].

***shadowRoot*.setHTML[p1227](*html*, *options*)**

Parses *html* using the HTML parser with options *options*, and replaces the children of *shadowRoot* with the result. *shadowRoot*'s host provides context for the HTML parser. The parsed fragment is sanitized[p1243] based on the *options*'s "sanitizer[p1224]" member, and unsafe content is removed[p1240].

***element*.setHTMLUnsafe[p1227](*html*, *options*)**

Parses *html* using the HTML parser with options *options*, and replaces the children of *element* with the result. *element* provides context for the HTML parser. If the *options* dictionary contains a "sanitizer[p1224]" member, it is used to sanitize[p1243] the parsed fragment before it is inserted into *element*. If the *options* dictionary's "runScripts[p1224]" member is true, scripts contained in *html* will be executed immediately after the node tree is updated.

***shadowRoot*.setHTMLUnsafe[p1227](*html*, *options*)**

Parses *html* using the HTML parser with options *options*, and replaces the children of *shadowRoot* with the result. *shadowRoot*'s host provides context for the HTML parser. If the *options* dictionary contains a "sanitizer[p1224]" member, it is used to sanitize[p1243] the parsed fragment before it is inserted into *shadowRoot*. If the *options* dictionary's "runScripts[p1224]" member is true, scripts contained in *html* will be executed immediately after the node tree is updated.

***doc* = Document.parseHTML[p1227](*html*, *options*)**

Parses *html* using the HTML parser with options *options*, and returns a new Document[p135] containing the result. The resulting document is sanitized[p1243] based on the *options*'s "sanitizer[p1224]" member, and unsafe content is removed[p1240].

***doc* = Document.parseHTMLUnsafe[p1227](*html*, *options*)**

Parses *html* using the HTML parser with options *options*, and returns the resulting Document[p135].

Note that script[p685] elements are not evaluated during parsing, and the resulting document's encoding will always be UTF-8. The document's URL will be about:blank[p54]. If the *options* dictionary contains a "sanitizer[p1224]" member, it is used to sanitize[p1243] the resulting DOM.

⚠Warning!

The methods with an `Unsafe` suffix perform no sanitization to remove potentially-dangerous elements and attributes like `script`[p685] or event handler content attributes[p1207].

`Element`'s **`setHTML(html, options)`** method steps are:

1. Let *target* be this's template contents[p707] if this is a `template`[p705] element; otherwise this.
2. Set and filter HTML[p1242] given *target*, *html*, *options*, and true.

`ShadowRoot`'s **`setHTML(html, options)`** method steps are:

1. Set and filter HTML[p1242] given this, *html*, *options*, and true.

`Element`'s **`setHTMLUnsafe(html, options)`** method steps are:

1. Let *compliantHTML* be the result of invoking the get trusted type compliant string algorithm with `TrustedHTML`, this's relevant global object[p1151], *html*, "`Element setHTMLUnsafe`", and "`script`".
2. Let *target* be this's template contents[p707] if this is a `template`[p705] element; otherwise this.
3. Set and filter HTML[p1242] given *target*, *compliantHTML*, *options*, and false.

`ShadowRoot`'s **`setHTMLUnsafe(html, options)`** method steps are:

1. Let *compliantHTML* be the result of invoking the get trusted type compliant string algorithm with `TrustedHTML`, this's relevant global object[p1151], *html*, "`ShadowRoot setHTMLUnsafe`", and "`script`".
2. Set and filter HTML[p1242] given this, *compliantHTML*, *options*, and false.

The static **`parseHTML(html, options)`** method steps are:

1. Let *document* be a new `Document`[p135], whose content type is "`text/html`" and origin is this's relevant global object[p1151]'s associated `Document`[p966]'s origin.

   Note

   Since document *does not have a browsing context[p1046], scripting is disabled[p1152].*

2. Set *document*'s allow declarative shadow roots to true.
3. Parse HTML from a string[p1226] given *document* and *html*.
4. Let *sanitizer* be the result of calling get a sanitizer instance from options[p1243] with *options* and true.
5. Call sanitize[p1243] on *document* with *sanitizer* and true.
6. Return *document*.

The static **`parseHTMLUnsafe(html, options)`** method steps are:

1. Let *compliantHTML* be the result of invoking the get trusted type compliant string algorithm with `TrustedHTML`, the current global object[p1151], *html*, "`Document parseHTMLUnsafe`", and "`script`".
2. Let *document* be a new `Document`[p135], whose content type is "`text/html`" and origin is this's relevant global object[p1151]'s associated `Document`[p966]'s origin.

   Note

   Since document *does not have a browsing context[p1046], scripting is disabled[p1152].*

3. Set *document*'s allow declarative shadow roots to true.
4. Parse HTML from a string[p1226] given *document* and *compliantHTML*.
5. Let *sanitizer* be the result of calling get a sanitizer instance from options[p1243] with *options* and false.
6. Call sanitize[p1243] on *document* with *sanitizer* and false.

7. Return *document*.

8.5.3 HTML serialization methods §[p1228]

For web developers (non-normative)

***html* = *element*.getHTML[p1228]({ serializableShadowRoots[p1224], shadowRoots[p1224] })**

Returns the result of serializing *element* to HTML. Shadow roots within *element* are serialized according to the provided options:

- If serializableShadowRoots[p1224] is true, then all shadow roots marked as serializable are serialized.
- If the shadowRoots[p1224] array is provided, then all shadow roots specified in the array are serialized, regardless of whether or not they are marked as serializable.

If neither option is provided, then no shadow roots are serialized.

***html* = *shadowRoot*.getHTML[p1228]({ serializableShadowRoots[p1224], shadowRoots[p1224] })**

Returns the result of serializing *shadowRoot* to HTML, using its shadow host as the context element. Shadow roots within *shadowRoot* are serialized according to the provided options, as above.

Element's **getHTML(*options*)** method steps are to return the result of HTML fragment serialization algorithm[p1468] with this, *options*["serializableShadowRoots[p1224]"], and *options*["shadowRoots[p1224]"].

ShadowRoot's **getHTML(*options*)** method steps are to return the result of HTML fragment serialization algorithm[p1468] with this, *options*["serializableShadowRoots[p1224]"], and *options*["shadowRoots[p1224]"].

8.5.4 The innerHTML[p1229] property §[p1228]

The innerHTML[p1229] property has a number of outstanding issues in the *DOM Parsing and Serialization* issue tracker, documenting various problems with its specification.

For web developers (non-normative)

***element*.innerHTML[p1229]**

Returns a fragment of HTML or XML that represents the element's contents.

In the case of an XML document, throws an "InvalidStateError" DOMException if the element cannot be serialized to XML.

element*.innerHTML[p1229] = *value

Replaces the contents of the element with nodes parsed from the given string.

In the case of an XML document, throws a "SyntaxError" DOMException if the given string is not well-formed.

***shadowRoot*.innerHTML[p1229]**

Returns a fragment of HTML that represents the shadow roots's contents.

shadowRoot*.innerHTML[p1229] = *value

Replaces the contents of the shadow root with nodes parsed from the given string.

⚠Warning!

These properties' setters perform no sanitization to remove potentially-dangerous elements and attributes like script[p685] or event handler content attributes[p1207].

The **fragment serializing algorithm steps**, given an Element, Document[p135], or DocumentFragment *node* and a boolean *require well-formed*, are:

1. Let *context document* be *node*'s node document.
2. If *context document* is an HTML document, return the result of HTML fragment serialization algorithm[p1468] with *node*, false, and « ».

3. Return the XML serialization of *node* given *require well-formed*.

The **fragment parsing algorithm steps**, given an `Element` or `DocumentFragment` *target*, a string *markup*, and an optional parser scripting mode[p1387] *scriptingMode* (default Inert[p1387]), are:

1. Assert: *scriptingMode* is either Inert[p1387] or Fragment[p1387].
2. If *target*'s node document is an XML document, then return the result of invoking the XML fragment parsing algorithm[p1487] given *target* and *markup*.
3. Return the result of the HTML fragment parsing algorithm[p1473] given *target*, *markup*, false, and *scriptingMode*.

`Element`'s **`innerHTML`** getter steps are to return the result of running fragment serializing algorithm steps[p1228] with this and true.

`ShadowRoot`'s **`innerHTML`** getter steps are to return the result of running fragment serializing algorithm steps[p1228] with this and true.

`Element`'s `innerHTML`[p1229] setter steps are:

1. Let *compliantString* be the result of invoking the get trusted type compliant string algorithm with `TrustedHTML`, this's relevant global object[p1151], the given value, "`Element innerHTML`", and "`script`".
2. Let *target* be this.
3. If *target* is a `template`[p705] element, then set *target* to the `template`[p705] element's template contents[p707] (a `DocumentFragment`).

   Note

   Setting `innerHTML`[p1229] on a `template`[p705] element will replace all the nodes in its template contents[p707] rather than its children.
4. Let *fragment* be the result of invoking the fragment parsing algorithm steps[p1229] with *target* and *compliantString*.
5. Replace all with *fragment* within *target*.

`ShadowRoot`'s `innerHTML`[p1229] setter steps are:

1. Let *compliantString* be the result of invoking the get trusted type compliant string algorithm with `TrustedHTML`, this's relevant global object[p1151], the given value, "`ShadowRoot innerHTML`", and "`script`".
2. Let *fragment* be the result of invoking the fragment parsing algorithm steps[p1229] with this and *compliantString*.
3. Replace all with *fragment* within this.

8.5.5 The `outerHTML`[p1230] property §[p1229]

The `outerHTML`[p1230] property has a number of outstanding issues in the *DOM Parsing and Serialization* issue tracker, documenting various problems with its specification.

For web developers (non-normative)

***element*.outerHTML**[p1230]

Returns a fragment of HTML or XML that represents the element and its contents.

In the case of an XML document, throws an "`InvalidStateError`" `DOMException` if the element cannot be serialized to XML.

element*.outerHTML**[p1230] **= *value

Replaces the element with nodes parsed from the given string.

In the case of an XML document, throws a "`SyntaxError`" `DOMException` if the given string is not well-formed.

Throws a "`NoModificationAllowedError`" `DOMException` if the parent of the element is a `Document`[p135].

⚠Warning!

This property's setter performs no sanitization to remove potentially-dangerous elements and attributes like

script[p685] ***or event handler content attributes***[p1207]***.***

Element's **outerHTML** getter steps are:

1. Let *element* be a fictional node whose only child is this.
2. Return the result of running fragment serializing algorithm steps[p1228] with *element* and true.

Element's outerHTML[p1230] setter steps are:

1. Let *compliantString* be the result of invoking the get trusted type compliant string algorithm with TrustedHTML, this's relevant global object[p1151], the given value, "Element outerHTML", and "script".
2. Let *parent* be this's parent.
3. If *parent* is null, return. There would be no way to obtain a reference to the nodes created even if the remaining steps were run.
4. If *parent* is a Document[p135], throw a "NoModificationAllowedError" DOMException.
5. If *parent* is a DocumentFragment, set *parent* to the result of creating an element given this's node document, "body", and the HTML namespace.
6. Let *fragment* be the result of invoking the fragment parsing algorithm steps[p1229] given *parent* and *compliantString*.
7. Replace this with *fragment* within this's parent.

8.5.6 The insertAdjacentHTML()[p1230] method §[p1230]

The insertAdjacentHTML()[p1230] method has a number of outstanding issues in the *DOM Parsing and Serialization* issue tracker, documenting various problems with its specification.

For web developers (non-normative)

***element*.insertAdjacentHTML**[p1230]**(*position*, *string*)**

Parses *string* as HTML or XML and inserts the resulting nodes into the tree in the position given by the *position* argument, as follows:

"beforebegin"
Before the element itself (i.e., after *element*'s previous sibling)

"afterbegin"
Just inside the element, before its first child.

"beforeend"
Just inside the element, after its last child.

"afterend"
After the element itself (i.e., before *element*'s next sibling)

Throws a "SyntaxError" DOMException if the arguments have invalid values (e.g., in the case of an XML document, if the given string is not well-formed).

Throws a "NoModificationAllowedError" DOMException if the given position isn't possible (e.g. inserting elements after the root element of a Document[p135]).

⚠Warning!

This method performs no sanitization to remove potentially-dangerous elements and attributes like script[p685] ***or event handler content attributes***[p1207]***.***

Element's **insertAdjacentHTML(*position*, *string*)** method steps are:

1. Let *compliantString* be the result of invoking the get trusted type compliant string algorithm with TrustedHTML, this's relevant global object[p1151], *string*, "Element insertAdjacentHTML", and "script".

2. Let *context* be null.

3. Use the first matching item from this list:

   ↪ **If *position* is an ASCII case-insensitive match for the string "beforebegin"**

   ↪ **If *position* is an ASCII case-insensitive match for the string "afterend"**

   1. Set *context* to this's parent[p974].
   2. If *context* is null or a Document[p135], throw a "NoModificationAllowedError" DOMException.

   ↪ **If *position* is an ASCII case-insensitive match for the string "afterbegin"**

   ↪ **If *position* is an ASCII case-insensitive match for the string "beforeend"**

   Set *context* to this.

   ↪ **Otherwise**

   Throw a "SyntaxError" DOMException.

4. If *context* is not an Element or all of the following are true:

   - *context*'s node document is an HTML document;
   - *context*'s local name is "html[p179]"; and
   - *context*'s namespace is the HTML namespace,

   then set *context* to the result of creating an element given this's node document, "body", and the HTML namespace.

5. Let *fragment* be the result of invoking the fragment parsing algorithm steps[p1229] with *context* and *compliantString*.

6. Use the first matching item from this list:

   ↪ **If *position* is an ASCII case-insensitive match for the string "beforebegin"**

   Insert *fragment* into this's parent[p974] before this.

   ↪ **If *position* is an ASCII case-insensitive match for the string "afterbegin"**

   Insert *fragment* into this before its first child.

   ↪ **If *position* is an ASCII case-insensitive match for the string "beforeend"**

   Append *fragment* to this.

   ↪ **If *position* is an ASCII case-insensitive match for the string "afterend"**

   Insert *fragment* into this's parent[p974] before this's next sibling.

Note

As with other direct Node-manipulation APIs (and unlike innerHTML[p1229]), insertAdjacentHTML()[p1230] does not include any special handling for template[p705] elements. In most cases you will want to use templateEl.content[p708].insertAdjacentHTML()[p1230] instead of directly manipulating the child nodes of a template[p705] element.

8.5.7 The createContextualFragment()[p1232] method §[p12 31]

The createContextualFragment()[p1232] method has a number of outstanding issues in the *DOM Parsing and Serialization* issue tracker, documenting various problems with its specification.

For web developers (non-normative)

***docFragment* = *range*.createContextualFragment[p1232](*string*)**

Returns a DocumentFragment created from the markup string *string* using *range*'s start node as the context in which *fragment* is parsed.

⚠Warning!

This method performs no sanitization to remove potentially-dangerous elements and attributes like script[p685] or event handler content attributes[p1207].

IDL

```
partial interface Range {
  [CEReactions, NewObject] DocumentFragment createContextualFragment((TrustedHTML or DOMString) string);
};
```

Range's **createContextualFragment(*string*)** method steps are:

1. Let *compliantString* be the result of invoking the get trusted type compliant string algorithm with TrustedHTML, this's relevant global object[p1151], *string*, "Range createContextualFragment", and "script".
2. Let *node* be this's start node.
3. Let *element* be null.
4. If *node* implements Element, set *element* to *node*.
5. Otherwise, if *node* implements Text or Comment, set *element* to *node*'s parent element.
6. If *element* is null or all of the following are true:
   - *element*'s node document is an HTML document;
   - *element*'s local name is "html[p179]"; and
   - *element*'s namespace is the HTML namespace,

   then set *element* to the result of creating an element given this's node document, "body", and the HTML namespace.
7. Return the result of invoking the fragment parsing algorithm steps[p1229] with *element*, *compliantString*, and Fragment[p1387].

8.5.8 The XMLSerializer[p1232] interface §[p12 32]

The XMLSerializer[p1232] interface has a number of outstanding issues in the *DOM Parsing and Serialization* issue tracker, documenting various problems with its specification. The remainder of *DOM Parsing and Serialization* will be gradually upstreamed to this specification.

For web developers (non-normative)

***xmlSerializer* = new XMLSerializer[p1232]()**

Constructs a new XMLSerializer[p1232] object.

***string* = *xmlSerializer*.serializeToString[p1233](*root*)**

Returns the result of serializing *root* to XML.

Throws an "InvalidStateError" DOMException if *root* cannot be serialized to XML.

Note

The design of XMLSerializer[p1232], as a class that needs to be constructed and then have its serializeToString()[p1233] method called, is an unfortunate historical artifact. If we were designing this functionality today it would be a standalone function.

IDL

```
[Exposed=Window]
interface XMLSerializer {
  constructor();

  DOMString serializeToString(Node root);
};
```

The **new XMLSerializer()** constructor steps are to do nothing.

The **serializeToString(*root*)** method steps are:

1. Return the XML serialization of *root* given false.

8.6 HTML sanitization §p12 33

8.6.1 Introduction §p12 33

This section is non-normative.

Web applications often need to process untrusted HTML strings, such as when rendering user-generated content or using client-side templates. Safely inserting these strings into the DOM requires careful sanitization to prevent DOM-based cross-site scripting (XSS) attacks.

HTML sanitization provides a native mechanism for safely parsing and sanitizing HTML strings. By using the user agent's own HTML parser, they ensure the sanitized output accurately reflects how the browser will render the content, preventing script execution and mitigating advanced attacks such as script gadgets.

These APIs offer functionality to parse a string containing HTML into a DOM tree, and to filter the resulting tree according to a user-supplied configuration. The methods come in two main flavors: "safe" and "unsafe".

8.6.1.1 Safe and unsafe §p12 33

The "safe" methods will not generate any markup that executes script. That is, they are intended to be safe from XSS. The "unsafe" methods will parse and filter based on the provided configuration, but do not have the same safety guarantees by default.

8.6.2 The Sanitizer[p1233] interface §p12 33

IDL

```
[Exposed=Window]
interface Sanitizer {
  constructor(optional (SanitizerConfig or SanitizerPresets) configuration = "default");

  // Query configuration:
  SanitizerConfig get();

  // Modify a Sanitizer's lists and fields:
  boolean allowElement(SanitizerElementWithAttributes element);
  boolean removeElement(SanitizerElement element);
  boolean replaceElementWithChildren(SanitizerElement element);
  boolean allowProcessingInstruction(SanitizerPI pi);
  boolean removeProcessingInstruction(SanitizerPI pi);
  boolean allowAttribute(SanitizerAttribute attribute);
  boolean removeAttribute(SanitizerAttribute attribute);
  boolean setComments(boolean allow);
  boolean setDataAttributes(boolean allow);
  boolean setJavascriptURLs(boolean allow);

  // Remove markup that executes script.
  boolean removeUnsafe();
};
```

For web developers (non-normative)

***config* = *sanitizer*.get[p1236]()**

Returns a copy of the sanitizer's configuration[p1240].

***sanitizer*.allowElement(*element*)**
Ensures that the sanitizer configuration allows the specified element.

***sanitizer*.removeElement(*element*)**
Ensures that the sanitizer configuration blocks the specified element.

***sanitizer*.replaceElementWithChildren(*element*)**
Configures the sanitizer to remove the specified element but keep its child nodes.

***sanitizer*.allowAttribute(*attribute*)**
Configures the sanitizer to allow the specified attribute globally.

***sanitizer*.removeAttribute(*attribute*)**
Configures the sanitizer to block the specified attribute globally.

***sanitizer*.allowProcessingInstruction(*pi*)**
Configures the sanitizer to allow the specified processing instruction.

***sanitizer*.removeProcessingInstruction(*pi*)**
Configures the sanitizer to block the specified processing instruction.

***sanitizer*.setComments(*allow*)**
Sets whether the sanitizer preserves comments.

***sanitizer*.setDataAttributes(*allow*)**
Sets whether the sanitizer preserves custom data attributes (e.g., `data-*`).

***sanitizer*.setJavascriptURLs(*allow*)**
Sets whether the sanitizer preserves attributes containing `javascript:` URLs.

***sanitizer*.removeUnsafe()**
Modifies the configuration to automatically remove elements and attributes that are considered unsafe.

A `Sanitizer` object has an associated **configuration**, which is a SanitizerConfig.

The **`new Sanitizer(`*`configuration`*`)`** constructor steps are:

1. If *configuration* is a SanitizerPresets string:
    1. Assert: *configuration* is "default".
    2. Set *configuration* to the built-in safe default configuration.
2. Configure this given *configuration* and true.

To **configure** a `Sanitizer` *sanitizer*, given a dictionary *configuration* and a boolean *permissiveDefaults*:

1. Canonicalize the configuration *configuration* with *permissiveDefaults*.
2. If *configuration* is not valid, then throw a `TypeError`.
3. Set *sanitizer*'s configuration to *configuration*.

To **canonicalize the configuration** SanitizerConfig *configuration* with a boolean *permissiveDefaults*:

1. If neither *configuration*["`elements`"] nor *configuration*["`removeElements`"] exists, then set *configuration*["`removeElements`"] to an empty list.
2. If neither *configuration*["`attributes`"] nor *configuration*["`removeAttributes`"] exists, then set *configuration*["`removeAttributes`"] to an empty list.
3. If neither *configuration*["`processingInstructions`"] nor *configuration*["`removeProcessingInstructions`"] exists:
    1. If *permissiveDefaults* is true, then set *configuration*["`removeProcessingInstructions`"] to an empty list.
    2. Otherwise, set *configuration*["`processingInstructions`"] to an empty list.

4. If *configuration*["elements"] exists:

    1. Let *newElements* be « ».

    2. For each *element* of *configuration*["elements"], append the result of canonicalizing *element* to *newElements*.

    3. Set *configuration*["elements"] to *newElements*.

5. If *configuration*["removeElements"] exists, then set *configuration*["removeElements"] to the result of canonicalizing *configuration*["removeElements"].

6. If *configuration*["attributes"] exists, then set *configuration*["attributes"] to the result of canonicalizing *configuration*["attributes"].

7. If *configuration*["removeAttributes"] exists, then set *configuration*["removeAttributes"] to the result of canonicalizing *configuration*["removeAttributes"].

8. If *configuration*["replaceWithChildrenElements"] exists, then set *configuration*["replaceWithChildrenElements"] to the result of canonicalizing *configuration*["replaceWithChildrenElements"].

9. If *configuration*["processingInstructions"] exists, then set *configuration*["processingInstructions"] to the result of canonicalizing *configuration*["processingInstructions"].

10. If *configuration*["removeProcessingInstructions"] exists, then set *configuration*["removeProcessingInstructions"] to the result of canonicalizing *configuration*["removeProcessingInstructions"].

11. If *configuration*["comments"] does not exist, then set it to *permissiveDefaults*.

12. If *configuration*["attributes"] exists and *configuration*["dataAttributes"] does not exist, then set it to *permissiveDefaults*.

13. If *configuration*["javascriptURLs"] does not exist, then set it to *permissiveDefaults*.

To **canonicalize a sanitizer list** *list*:

1. Let *newList* be « ».

2. For each *item* in *list*, append the result of canonicalizing *item* to *newList*.

3. Return *newList*.

To **canonicalize a processing instruction list** *list*:

1. Let *newList* be « ».

2. For each *item* in *list*, append the result of canonicalizing *item* to *newList*.

3. Return *newList*.

To **canonicalize a processing instruction** given a SanitizerPI *pi*:

1. If *pi* is a DOMString, then return «["target" → *pi*]».

2. Assert: *pi* is a dictionary and *pi*["target"] exists.

3. Return «["target" → *pi*["target"]]».

To **canonicalize a sanitizer name** given a DOMString or dictionary *name*, and a default namespace *defaultNamespace* (default null):

1. If *name* is a DOMString, then return «["name" → *name*, "namespace" → *defaultNamespace*]».

2. Assert: *name* is a dictionary and both *name*["name"] and *name*["namespace"] exist.

3. If *name*["namespace"] is the empty string, then set it to null.

4. Return «["name" → *name*["name"], "namespace" → *name*["namespace"]]».

To **canonicalize a sanitizer element** given a SanitizerElement *element*:

1. Return the result of canonicalizing[p1235] *element* with the HTML namespace as the default namespace.

To **canonicalize a sanitizer element list** *list*:

1. Let *newList* be « ».
2. For each *item* in *list*, append the result of canonicalizing[p1235] *item* to *newList*.
3. Return *newList*.

To find the **canonicalized intersection** of lists *A* and *B*:

1. Let *setA* be « ».
2. Let *setB* be « ».
3. For each *entry* of *A*, append the result of canonicalizing[p1235] *entry* to *setA*.
4. For each *entry* of *B*, append the result of canonicalizing[p1235] *entry* to *setB*.
5. Return the intersection of *setA* and *setB*.

The **`get()`** method steps are:

Note

Outside of the `get()`[p1236] method, the order of the `Sanitizer`[p1233]'s elements and attributes is unobservable. By explicitly sorting the result of this method, we give implementations the opportunity to optimize by, for example, using unordered sets internally.

1. Let *config* be this's configuration[p1234].
2. Assert: *config* is valid[p1247].
3. If *config*["`elements`[p1240]"] exists:
    1. For each *element* of *config*["`elements`[p1240]"]:
        1. If *element*["`attributes`[p1240]"] exists, then set *element*["`attributes`[p1240]"] to the result of sorting *element*["`attributes`[p1240]"], with compare sanitizer items[p1246].
        2. If *element*["`removeAttributes`[p1240]"] exists, then set *element*["`removeAttributes`[p1240]"] to the result of sorting *element*["`removeAttributes`[p1240]"], with compare sanitizer items[p1246].
    2. Set *config*["`elements`[p1240]"] to the result of sorting *config*["`elements`[p1240]"], with compare sanitizer items[p1246].
4. Otherwise:
    1. Set *config*["`removeElements`[p1240]"] to the result of sorting *config*["`removeElements`[p1240]"], with compare sanitizer items[p1246].
5. If *config*["`replaceWithChildrenElements`[p1240]"] exists, then set *config*["`replaceWithChildrenElements`[p1240]"] to the result of sorting *config*["`replaceWithChildrenElements`[p1240]"], with compare sanitizer items[p1246].
6. If *config*["`processingInstructions`[p1240]"] exists, then set *config*["`processingInstructions`[p1240]"] to the result of sorting *config*["`processingInstructions`[p1240]"], with *piA*["`target`[p1240]"] being code unit less than *piB*["`target`[p1240]"].
7. Otherwise:
    1. Set *config*["`removeProcessingInstructions`[p1241]"] to the result of sorting *config*["`removeProcessingInstructions`[p1241]"], with *piA*["`target`[p1240]"] being code unit less than *piB*["`target`[p1240]"].
8. If *config*["`attributes`[p1241]"] exists, then set *config*["`attributes`[p1241]"] to the result of sorting *config*["`attributes`[p1241]"] given compare sanitizer items[p1246].
9. Otherwise:
    1. Set *config*["`removeAttributes`[p1241]"] to the result of sorting *config*["`removeAttributes`[p1241]"] given compare sanitizer items[p1246].
10. Return *config*.

The **allowElement(*element*)** method steps are:

1. Let *configuration* be this's configuration[p1234].
2. Assert: *configuration* is valid[p1247].
3. Set *element* to the result of canonicalizing[p1247] *element*.
4. If *configuration*["elements[p1240]"] exists:
    1. Let *modified* be the result of removing *element* from *configuration*["replaceWithChildrenElements[p1240]"].
    2. If *configuration*["attributes[p1241]"] exists:
        1. If *element*["attributes[p1240]"] exists:
            1. Set *element*["attributes[p1240]"] to the result of creating a set from *element*["attributes[p1240]"].
            2. Set *element*["attributes[p1240]"] to the difference of *element*["attributes[p1240]"] and *configuration*["attributes[p1241]"].
            3. If *configuration*["dataAttributes[p1241]"] is true, then remove all items *item* from *element*["attributes[p1240]"] where *item* is a custom data attribute[p171].
        2. If *element*["removeAttributes[p1240]"] exists:
            1. Set *element*["removeAttributes[p1240]"] to the result of creating a set from *element*["removeAttributes[p1240]"].
            2. Set *element*["removeAttributes[p1240]"] to the intersection[p1236] of *element*["removeAttributes[p1240]"] and *configuration*["attributes[p1241]"].
    3. Otherwise:
        1. If *element*["attributes[p1240]"] exists:
            1. Set *element*["attributes[p1240]"] to the result of creating a set from *element*["attributes[p1240]"].
            2. Set *element*["attributes[p1240]"] to the difference of *element*["attributes[p1240]"] and *element*["removeAttributes[p1240]"] with default « ».
            3. Remove *element*["removeAttributes[p1240]"].
            4. Set *element*["attributes[p1240]"] to the difference of *element*["attributes[p1240]"] and *configuration*["removeAttributes[p1241]"].
        2. If *element*["removeAttributes[p1240]"] exists:
            1. Set *element*["removeAttributes[p1240]"] to the result of creating a set from *element*["removeAttributes[p1240]"].
            2. Set *element*["removeAttributes[p1240]"] to the difference of *element*["removeAttributes[p1240]"] and *configuration*["removeAttributes[p1241]"].
    4. If *configuration*["elements[p1240]"] does not contain *element*:
        1. Append *element* to *configuration*["elements[p1240]"].
        2. Return true.
    5. Let *currentElement* be the item in *configuration*["elements[p1240]"] whose name[p1240] member is *element*'s name[p1240] member and whose namespace[p1240] member is *element*'s namespace[p1240] member.
    6. If *element* is equal to *currentElement*, then return *modified*.
    7. Remove *element* from *configuration*["elements[p1240]"].
    8. Append *element* to *configuration*["elements[p1240]"].
    9. Return true.

5. Otherwise:

    1. If *element*["attributes"] exists or *element*["removeAttributes"] with default « » is not empty, then return false.

    2. Let *modified* be the result of removing *element* from *configuration*["replaceWithChildrenElements"].

    3. If *configuration*["removeElements"] does not contain *element*, then return *modified*.

    4. Remove *element* from *configuration*["removeElements"].

    5. Return true.

The **removeElement(*element*)** method steps are to return the result of removing *element* from this's configuration.

The **replaceElementWithChildren(*element*)** method steps are:

1. Let *configuration* be this's configuration.
2. Assert: *configuration* is valid.
3. Set *element* to the result of canonicalizing *element*.
4. If the built-in non-replaceable elements list contains *element*, then return false.
5. Let *modified* be the result of removing *element* from *configuration*["elements"].
6. If removing *element* from *configuration*["removeElements"] is true, then set *modified* to true.
7. If *configuration*["replaceWithChildrenElements"] does not contain *element*:

    1. Append *element* to *configuration*["replaceWithChildrenElements"].

    2. Return true.

8. Return *modified*.

The **allowAttribute(*attribute*)** method steps are:

1. Let *configuration* be this's configuration.
2. Assert: *configuration* is valid.
3. Set *attribute* to the result of canonicalizing *attribute*.
4. If *configuration*["attributes"] exists:

    1. If *configuration*["dataAttributes"] is true and *attribute* is a custom data attribute, then return false.

    2. If *configuration*["attributes"] contains *attribute*, then return false.

    3. If *configuration*["elements"] exists:

        1. For each *element* in *configuration*["elements"]:

            1. If *element*["attributes"] with default « » contains *attribute*, then remove *attribute* from *element*["attributes"].

    4. Append *attribute* to *configuration*["attributes"].

    5. Return true.

5. Otherwise:

    1. If *configuration*["removeAttributes"] does not contain *attribute*, then return false.

    2. Remove *attribute* from *configuration*["removeAttributes"].

    3. Return true.

The **removeAttribute(*attribute*)** method steps are to return the result of removing *attribute* from this's configuration.

The **`setComments(allow)`** method steps are:

1. Let *configuration* be this's configuration[p1234].
2. Assert: *configuration* is valid[p1247].
3. If *configuration*["comments[p1241]"] is *allow*, then return false.
4. Set *configuration*["comments[p1241]"] to *allow*.
5. Return true.

The **`setDataAttributes(allow)`** method steps are:

1. Let *configuration* be this's configuration[p1234].
2. Assert: *configuration* is valid[p1247].
3. If *configuration*["attributes[p1241]"] does not exist, then return false.
4. If *configuration*["dataAttributes[p1241]"] exists and is equal to *allow*, then return false.
5. If *allow* is true:
    1. If *configuration*["elements[p1240]"] exists:
        1. For each *element* of *configuration*["elements[p1240]"]:
            1. If *element*["attributes[p1240]"] exists, then remove all items *item* from *element*["attributes[p1240]"] where *item* is a custom data attribute[p171].
    2. Remove all items *item* from *configuration*["attributes[p1241]"] where *item* is a custom data attribute[p171].
6. Set *configuration*["dataAttributes[p1241]"] to *allow*.
7. Return true.

The **`setJavascriptURLs(allow)`** method steps are:

1. Let *configuration* be this's configuration[p1234].
2. Assert: *configuration* is valid[p1247].
3. If *configuration*["javascriptURLs[p1241]"] is *allow*, then return false.
4. Set *configuration*["javascriptURLs[p1241]"] to *allow*.
5. Return true.

The **`allowProcessingInstruction(pi)`** method steps are:

1. Let *configuration* be this's configuration[p1234].
2. Assert: *configuration* is valid[p1247].
3. Set *pi* to the result of canonicalizing[p1235] *pi*.
4. If *configuration*["processingInstructions[p1240]"] exists:
    1. If *configuration*["processingInstructions[p1240]"] contains *pi*, then return false.
    2. Append *pi* to *configuration*["processingInstructions[p1240]"].
    3. Return true.
5. Otherwise:
    1. If *configuration*["removeProcessingInstructions[p1241]"] contains *pi*:
        1. Remove *pi* from *configuration*["removeProcessingInstructions[p1241]"].
        2. Return true.

2. Return false.

The **`removeProcessingInstruction(pi)`** method steps are:

1. Let *configuration* be this's configuration[p1234].
2. Assert: *configuration* is valid[p1247].
3. Set *pi* to the result of canonicalizing[p1235] *pi*.
4. If *configuration*["`processingInstructions`[p1240]"] exists:
    1. If *configuration*["`processingInstructions`[p1240]"] contains *pi*:
        1. Remove *pi* from *configuration*["`processingInstructions`[p1240]"].
        2. Return true.
    2. Return false.
5. Otherwise:
    1. If *configuration*["`removeProcessingInstructions`[p1241]"] contains *pi*, then return false.
    2. Append *pi* to *configuration*["`removeProcessingInstructions`[p1241]"].
    3. Return true.

The **`removeUnsafe()`** method steps are to return the result of removing unsafe[p1246] from this's configuration[p1234].

8.6.3 Sanitizer configuration §[p1240]

IDL

```
dictionary SanitizerElementNamespace {
  required DOMString name;
  DOMString? _namespace = "http://www.w3.org/1999/xhtml";
};

// Used by "elements"
dictionary SanitizerElementNamespaceWithAttributes : SanitizerElementNamespace {
  sequence<SanitizerAttribute> attributes;
  sequence<SanitizerAttribute> removeAttributes;
};

dictionary SanitizerAttributeNamespace {
  required DOMString name;
  DOMString? _namespace = null;
};

dictionary SanitizerProcessingInstruction {
  required DOMString target;
};

typedef (DOMString or SanitizerElementNamespace) SanitizerElement;
typedef (DOMString or SanitizerElementNamespaceWithAttributes) SanitizerElementWithAttributes;
typedef (DOMString or SanitizerProcessingInstruction) SanitizerPI;
typedef (DOMString or SanitizerAttributeNamespace) SanitizerAttribute;

dictionary SanitizerConfig {
  sequence<SanitizerElementWithAttributes> elements;
  sequence<SanitizerElement> removeElements;
  sequence<SanitizerElement> replaceWithChildrenElements;

  sequence<SanitizerPI> processingInstructions;
```

```
  sequence<SanitizerPI> removeProcessingInstructions;

  sequence<SanitizerAttribute> attributes;
  sequence<SanitizerAttribute> removeAttributes;

  boolean comments;
  boolean dataAttributes;
  boolean javascriptURLs;
};
```

Note

SanitizerElementNamespace[p1240], SanitizerAttributeNamespace[p1240], SanitizerElementNamespaceWithAttributes[p1240], and SanitizerProcessingInstruction[p1240] dictionaries are equal when they have the same members and those are equal, per the equality of maps.

8.6.3.1 Configuration invariants §[p1241]

This section is non-normative.

Configurations can and ought to be modified by developers to suit their purposes. Options are to write a new SanitizerConfig[p1240] dictionary from scratch, to modify an existing Sanitizer[p1233]'s configuration by using the modifier methods, or to get()[p1236] an existing Sanitizer[p1233]'s configuration[p1234] as a dictionary and modify the dictionary and then create a new Sanitizer[p1233] with it.

An empty configuration allows everything (when called with the "unsafe" methods like setHTMLUnsafe()[p1227]). A configuration "**default**" contains a built-in safe default configuration[p1249]. Note that "safe" and "unsafe" sanitizer methods have different defaults.

Not all configuration dictionaries are valid. A valid configuration avoids redundancy (like specifying the same element to be allowed twice) and contradictions (like specifying an element to be both removed and allowed.)

Several conditions need to hold for a configuration to be valid:

- Mixing global allow- and remove-lists:
  - elements[p1240] or removeElements[p1240] can exist, but not both. If both are missing, this is equivalent to removeElements[p1240] being an empty list.
  - attributes[p1241] or removeAttributes[p1241] can exist, but not both. If both are missing, this is equivalent to removeAttributes[p1241] being an empty list.
  - dataAttributes[p1241] is conceptually an extension of the attributes[p1241] allow-list. The dataAttributes[p1241] member is only allowed when an attributes[p1241] list is used.
- Duplicate entries between different global lists:
  - There are no duplicate entries (i.e., no same elements) between elements[p1240], removeElements[p1240], or replaceWithChildrenElements[p1240].
  - There are no duplicate entries (i.e., no same attributes) between attributes[p1241] or removeAttributes[p1241].
- Mixing local allow- and remove-lists on the same element:
  - When an attributes[p1241] list exists, both, either or none of the attributes[p1240] and removeAttributes[p1240] lists are allowed on the same element.
  - When a removeAttributes[p1241] list exists, either or none of the attributes[p1240] and removeAttributes[p1240] lists are allowed on the same element, but not both.
- Duplicate entries on the same element:
  - There are no duplicate entries between attributes[p1240] and removeAttributes[p1240] on the same element.
- No element from the built-in non-replaceable elements list[p1251] appears in replaceWithChildrenElements[p1240], since replacing these elements with their children could lead to re-parsing issues or invalid node trees.

The `elements` element allow-list can also specify allowing or removing attributes for a given element. This is meant to mirror this standard's structure, which knows both global attributes as well as local attributes that apply to a specific element. Global and local attributes can be mixed, but note that ambiguous configurations where a particular attribute would be allowed by one list and forbidden by another, are generally invalid.

	global `attributes`	**global `removeAttributes`**
local `attributes`	An attribute is allowed if it matches either list. No duplicates are allowed.	An attribute is only allowed if it's in the local allow list. No duplicate entries between global remove and local allow lists are allowed. Note that the global remove list has no function for this particular element, but can apply to other elements that do not have a local allow list.
local `removeAttributes`	An attribute is allowed if it's in the global allow-list, but not in the local remove-list. Local remove has to be a subset of the global allow lists.	An attribute is allowed if it is in neither list. No duplicate entries between global remove and local remove lists are allowed.

Please note the asymmetry where mostly no duplicates between global and per-element lists are permitted, but in the case of a global allow-list and a per-element remove-list the latter has to be a subset of the former. An excerpt of the table above, only focusing on duplicates, is as follows:

	global `attributes`	**global `removeAttributes`**
local `attributes`	No duplicates are allowed.	No duplicates are allowed.
local `removeAttributes`	Local remove has to be a subset of the global allow lists.	No duplicates are allowed.

The `dataAttributes` setting allows custom data attributes. The rules above easily extends to custom data attributes if one considers `dataAttributes` to be an allow-list:

	global `attributes` and `dataAttributes` set
local `attributes`	All custom data attributes are allowed. No custom data attributes can be listed in any allow-list, as that would mean a duplicate entry.
local `removeAttributes`	A custom data attribute is allowed, unless it's listed in the local remove-list. No custom data attribute can be listed in the global allow-list, as that would mean a duplicate entry.

Putting these rules in words:

- Duplicates and interactions between global and local lists:
  - If a global `attributes` allow list exists, then all element's local lists:
    - If a local `attributes` allow list exists, there can be no duplicate entries between these lists.
    - If a local `removeAttributes` remove list exists, then all its entries also need to be listed in the global `attributes` allow list.
    - If `dataAttributes` is true, then no custom data attributes can be listed in any of the allow-lists.
  - If a global `removeAttributes` remove list exists:
    - If a local `attributes` allow list exists, there can be no duplicate entries between these lists.
    - If a local `removeAttributes` remove list exists, there can be no duplicate entries between these lists.
    - Not both a local `attributes` allow list and local `removeAttributes` remove list exists.
    - `dataAttributes` has to be false.

8.6.4 Sanitization algorithms §42

To **set and filter HTML**, given an `Element` or `DocumentFragment` *target*, a string *html*, a dictionary *options*, and a boolean *safe*:

1. Let *context* be *target* if *target* is an `Element`; otherwise *target*'s host.
2. Assert: *context* is non-null.
3. If all of the following are true:

- *safe*;
- *context*'s local name is "`script`[p685]"; and
- *context*'s namespace is the HTML namespace or the SVG namespace,

then return.

4. Let *sanitizer* be the result of calling getting a sanitizer[p1243] from *options* given *safe*.
5. Let *scriptingMode* be Inert[p1387].
6. If *options*["`runScripts`[p1224]"] is true:
    1. Assert: *safe* is false.
    2. Set *scriptingMode* to Fragment[p1387].
7. Let *fragment* be the result of invoking the HTML fragment parsing algorithm[p1473] given *target*, *html*, true, and *scriptingMode*.

> **Note**
> *Scripts in* fragment *will only execute once inserted into* target.

8. Sanitize[p1243] *fragment* given *sanitizer* and *safe*.
9. Replace all with *fragment* within *target*.

To **get a sanitizer instance from options** from a dictionary *options* with a boolean *safe*:

1. Let *sanitizerSpec* be "`default`[p1241]".
2. If *options*["`sanitizer`[p1224]"] exists, then set *sanitizerSpec* to *options*["`sanitizer`[p1224]"].
3. Assert: *sanitizerSpec* is either a `Sanitizer`[p1233] instance, a SanitizerPresets[p1224] member, or a SanitizerConfig[p1240] dictionary.
4. If *sanitizerSpec* is a string:
    1. Assert: *sanitizerSpec* is "`default`[p1241]".
    2. Set *sanitizerSpec* to the built-in safe default configuration[p1249].
5. If *sanitizerSpec* is a dictionary:
    1. Let *sanitizer* be a new `Sanitizer`[p1233] object.
    2. Let *permissiveDefaults* be true if *safe* is false; false otherwise.
    3. Configure[p1234] *sanitizer* given *sanitizerSpec* and *permissiveDefaults*.
    4. Set *sanitizerSpec* to *sanitizer*.
6. Return *sanitizerSpec*.

To **sanitize** a `Node` *node* with a `Sanitizer`[p1233] *sanitizer* and a boolean *safe*:

1. Let *configuration* be *sanitizer*'s configuration[p1234].
2. Assert: *configuration* is valid[p1247].
3. If *safe* is true, then remove unsafe[p1246] from *configuration*.
4. Run the inner sanitize steps[p1243] given *node* and *configuration*.

To perform the **inner sanitize steps** given a `Node` *node* and a SanitizerConfig[p1240] *configuration*:

1. For each *child* of *node*'s children:
    1. Assert: *child* is a `Text`, `Comment`, `Element`, `ProcessingInstruction`, or `DocumentType` node.
    2. If *child* is a `DocumentType` or `Text` node, then continue.
    3. If *child* is a `Comment` node:
        1. If *configuration*["`comments`[p1241]"] is not true, then remove *child*.

2. Continue.

4. If *child* is a `ProcessingInstruction` node:

    1. Let *piTarget* be *child*'s target.

    2. If *configuration*["`processingInstructions`[p1240]"] exists:

        1. If *configuration*["`processingInstructions`[p1240]"] does not contain *piTarget*, then remove *child*.

    3. Otherwise:

        1. If *configuration*["`removeProcessingInstructions`[p1241]"] contains *piTarget*, then remove *child*.

5. Otherwise:

    1. Let *elementName* be a SanitizerElementNamespace[p1240] with *child*'s local name and namespace.

    2. If *configuration*["`replaceWithChildrenElements`[p1240]"] exists and *configuration*["`replaceWithChildrenElements`[p1240]"] contains *elementName*:

        1. Assert: *node* is not a `Document`[p135].

        2. Run the inner sanitize steps[p1243] given *child* and *configuration*.

        3. Let *fragment* be the result of creating a document fragment given *node*'s node document.

        4. For each *innerChild* of *child*'s children, append *innerChild* to *fragment*.

        5. Replace *child* with *fragment* within *node*. Assert that this did not throw.

        6. Continue.

    3. If *configuration*["`elements`[p1240]"] exists:

        1. If *configuration*["`elements`[p1240]"] does not contain *elementName*, then remove *child* and continue.

    4. Otherwise:

        1. If *configuration*["`removeElements`[p1240]"] contains *elementName*, then remove *child* and continue.

    5. If *elementName*["`name`[p1240]"] is "`template`[p705]" and *elementName*["`namespace`[p1240]"] is the HTML namespace, then run the inner sanitize steps[p1243] given *child*'s template contents[p707] and *configuration*.

    6. If *child* is a shadow host, then run the inner sanitize steps[p1243] given *child*'s shadow root and *configuration*.

    7. Let *elementWithLocalAttributes* be «[ ]».

    8. If *configuration*["`elements`[p1240]"] exists and *configuration*["`elements`[p1240]"] contains *elementName*, then set *elementWithLocalAttributes* to *configuration*["`elements`[p1240]"][*elementName*].

    9. For each *attribute* in *child*'s attribute list:

        1. Let *attrName* be a SanitizerAttributeNamespace[p1240] with *attribute*'s local name and namespace.

        2. If *elementWithLocalAttributes*["`removeAttributes`[p1240]"] exists and *elementWithLocalAttributes*["`removeAttributes`[p1240]"] contains *attrName*, then remove *attribute*.

        3. Otherwise, if *configuration*["`attributes`[p1241]"] exists:

            1. If *configuration*["`attributes`[p1241]"] does not contain *attrName* and *elementWithLocalAttributes*["`attributes`[p1240]"] with default « » does not contain *attrName*, and if "`data-`" is not a code unit prefix of *attribute*'s local name or *attribute*'s namespace is not null or *configuration*["`dataAttributes`[p1241]"] is not true, then remove *attribute*.

4. Otherwise:

    1. If *elementWithLocalAttributes*["attributes"] exists and *elementWithLocalAttributes*["attributes"] does not contain *attrName*, then remove *attribute*.

    2. Otherwise, if *configuration*["removeAttributes"] contains *attrName*, then remove *attribute*.

5. If *configuration*["javascriptURLs"] is not true:

    1. If the pair (*elementName*, *attrName*) matches an entry in the built-in navigating URL attributes list, and if *attribute* contains a `javascript:` URL, then remove *attribute*.

    2. If *child*'s namespace is the MathML namespace, *attribute*'s local name is "`href`", *attribute*'s namespace is null or the XLink namespace, and *attribute* contains a `javascript:` URL, then remove *attribute*.

    3. If the built-in animating URL attributes list contains the pair (*elementName*, *attrName*), and *attribute*'s value is "`href`" or "`xlink:href`", then remove *attribute*.

10. Run the inner sanitize steps given *child* and *configuration*.

To determine whether an attribute *attribute* **contains a `javascript:` URL**:

1. Let *url* be the result of running the basic URL parser on *attribute*'s value.
2. If *url* is failure, then return false.
3. Return true if *url*'s scheme is "`javascript`", and false otherwise.

To **remove an element** *element* from a SanitizerConfig *configuration*:

1. Assert: *configuration* is valid.
2. Set *element* to the result of canonicalizing *element*.
3. Let *modified* be the result of removing *element* from *configuration*["replaceWithChildrenElements"].
4. If *configuration*["elements"] exists:
    1. If *configuration*["elements"] contains *element*:
        1. Remove *element* from *configuration*["elements"].
        2. Return true.
    2. Return *modified*.
5. Otherwise:
    1. If *configuration*["removeElements"] contains *element*, then return *modified*.
    2. Append *element* to *configuration*["removeElements"].
    3. Return true.

To **remove an attribute** *attribute* from a SanitizerConfig *configuration*:

1. Assert: *configuration* is valid.
2. Set *attribute* to the result of canonicalizing *attribute*.
3. If *configuration*["attributes"] exists:
    1. Let *modified* be the result of removing *attribute* from *configuration*["attributes"].
    2. If *configuration*["elements"] exists:
        1. For each *element* of *configuration*["elements"]:

1. If *element*["attributes"] with default « » contains *attribute*:
    1. Set *modified* to true.
    2. Remove *attribute* from *element*["attributes"].
2. If *element*["removeAttributes"] with default « » contains *attribute*:
    1. Assert: *modified* is true.
    2. Remove *attribute* from *element*["removeAttributes"].
3. Return *modified*.

4. Otherwise:
    1. If *configuration*["removeAttributes"] contains *attribute*, then return false.
    2. If *configuration*["elements"] exists:
        1. For each *element* in *configuration*["elements"]:
            1. If *element*["attributes"] with default « » contains *attribute*, then remove *attribute* from *element*["attributes"].
            2. If *element*["removeAttributes"] with default « » contains *attribute*, then remove *attribute* from *element*["removeAttributes"].
    3. Append *attribute* to *configuration*["removeAttributes"].
    4. Return true.

To **remove unsafe** from a SanitizerConfig *configuration*:

1. Assert: *configuration* is valid.
2. Let *result* be false.
3. For each *element* in built-in safe baseline configuration["removeElements"]:
    1. If removing *element* from *configuration* is true, then set *result* to true.
4. For each *attribute* in built-in safe baseline configuration["removeAttributes"]:
    1. If removing *attribute* from *configuration* is true, then set *result* to true.
5. For each *attribute* that is an event handler content attribute:
    1. If removing *attribute* from *configuration* is true, then set *result* to true.
6. If *configuration*["javascriptURLs"] is true:
    1. Set *result* to true.
    2. Set *configuration*["javascriptURLs"] to false.
7. Return *result*.

To **compare sanitizer items** *itemA* and *itemB*:

1. Let *namespaceA* be *itemA*["namespace"].
2. Let *namespaceB* be *itemB*["namespace"].
3. If *namespaceA* is null:
    1. If *namespaceB* is not null, then return true.
4. Otherwise:
    1. If *namespaceB* is null, then return false.

2. If *namespaceA* is code unit less than *namespaceB*, then return true.
3. If *namespaceA* is not *namespaceB*, then return false.

5. If *itemA*["name"] is code unit less than *itemB*["name"], then return true.
6. Return false.

To **canonicalize** a SanitizerElementWithAttributes *element*:

1. Let *result* be the result of canonicalizing *element*.
2. If *element* is a dictionary:
    1. If *element*["attributes"] exists, then set *result*["attributes"] to the result of canonicalizing *element*["attributes"].
    2. If *element*["removeAttributes"] exists, then set *result*["removeAttributes"] to the result of canonicalizing *element*["removeAttributes"].
3. If neither *result*["attributes"] nor *result*["removeAttributes"] exists, then set *result*["removeAttributes"] to an empty list.
4. Return *result*.

To determine whether a canonical SanitizerConfig *config* is **valid**:

Note

It's expected that the configuration being passed in has previously been run through the canonicalize the configuration steps. We will simply assert conditions that that algorithm is guaranteed to hold.

1. Assert: *config*["elements"] exists or *config*["removeElements"] exists.
2. If *config*["elements"] exists and *config*["removeElements"] exists, then return false.
3. Assert: Either *config*["processingInstructions"] exists or *config*["removeProcessingInstructions"] exists.
4. If *config*["processingInstructions"] exists and *config*["removeProcessingInstructions"] exists, then return false.
5. Assert: Either *config*["attributes"] exists or *config*["removeAttributes"] exists.
6. If *config*["attributes"] exists and *config*["removeAttributes"] exists, then return false.
7. Assert: All SanitizerElementNamespaceWithAttributes, SanitizerElementNamespace, SanitizerProcessingInstruction, and SanitizerAttributeNamespace items in *config* are canonical, meaning they have been run through canonicalizing, as appropriate.
8. If *config*["elements"] exists:
    1. If *config*["elements"] has duplicates, then return false.
9. Otherwise:
    1. If *config*["removeElements"] has duplicates, then return false.
10. If *config*["replaceWithChildrenElements"] exists and has duplicates, then return false.
11. If *config*["processingInstructions"] exists:
    1. If *config*["processingInstructions"] has duplicates, then return false.
12. Otherwise:
    1. If *config*["removeProcessingInstructions"] has duplicates, then return false.
13. If *config*["attributes"] exists:
    1. If *config*["attributes"] has duplicates, then return false.
14. Otherwise:

1. If *config*["removeAttributes"] has duplicates, then return false.

15. If *config*["replaceWithChildrenElements"] exists:

    1. For each *element* of *config*["replaceWithChildrenElements"]:

        1. If the built-in non-replaceable elements list contains *element*, then return false.

    2. If *config*["elements"] exists:

        1. If the intersection of *config*["elements"] and *config*["replaceWithChildrenElements"] is not empty, then return false.

    3. Otherwise:

        1. If the intersection of *config*["removeElements"] and *config*["replaceWithChildrenElements"] is not empty, then return false.

16. If *config*["attributes"] exists:

    1. Assert: *config*["dataAttributes"] exists.

    2. If *config*["elements"] exists:

        1. For each *element* of *config*["elements"]:

            1. If *element*["attributes"] exists and *element*["attributes"] has duplicates, then return false.

            2. If *element*["removeAttributes"] exists and *element*["removeAttributes"] has duplicates, then return false.

            3. If the intersection of *config*["attributes"] and *element*["attributes"] with default « » is not empty, then return false.

            4. If *element*["removeAttributes"] with default « » is not a subset of *config*["attributes"], then return false.

            5. If *config*["dataAttributes"] is true and *element*["attributes"] contains a custom data attribute, then return false.

    3. If *config*["dataAttributes"] is true and *config*["attributes"] contains a custom data attribute, then return false.

17. Otherwise:

    1. If *config*["elements"] exists:

        1. For each *element* of *config*["elements"]:

            1. If *element*["attributes"] exists and *element*["removeAttributes"] exists, then return false.

            2. If *element*["attributes"] exists and *element*["attributes"] has duplicates, then return false.

            3. If *element*["removeAttributes"] exists and *element*["removeAttributes"] has duplicates, then return false.

            4. If the intersection of *config*["removeAttributes"] and *element*["attributes"] with default « » is not empty, then return false.

            5. If the intersection of *config*["removeAttributes"] and *element*["removeAttributes"] with default « » is not empty, then return false.

    2. If *config*["dataAttributes"] exists, then return false.

18. Return true.

8.6.5 Sanitization constants §[p12 49]

An element's **sanitization category** can be one of the following:

Default

The element is included in the default SanitizerConfig[p1240].

Unsafe

The element can result in XSS. (Such elements are preserved by setHTMLUnsafe()[p1227] or parseHTMLUnsafe()[p1227], and removed by setHTML()[p1227], parseHTML()[p1227], and removeUnsafe()[p1240].)

Uncategorized

The element is neither removed nor included by default, but can still be added to a SanitizerConfig[p1240].

The **built-in safe baseline configuration** is a SanitizerConfig[p1240]. Its removeElements[p1240] list consists of all HTML elements normatively marked as Unsafe[p1249] within their individual definitions, along with the obsolete frame[p1537] element, and the SVG script and SVG use elements, and its removeAttributes[p1241] list is empty.

Note

Event handler content attributes are automatically removed by the remove unsafe[p1246] algorithm during safe sanitization, so the effective baseline behaves as if they were included in the removeAttributes[p1241] list.

The **built-in safe default configuration** is a SanitizerConfig[p1240] whose members are initialized as follows:

processingInstructions[p1240]

« »

attributes[p1241]

A list consisting of:

- dir[p167]
- lang[p165]
- title[p164]
- alignment-baseline
- baseline-shift
- clip-path
- clip-rule
- color
- color-interpolation
- cursor
- direction
- display
- displaystyle
- dominant-baseline
- fill
- fill-opacity
- fill-rule
- font-family
- font-size
- font-size-adjust
- font-stretch
- font-style
- font-variant
- font-weight
- letter-spacing
- marker-end
- marker-mid
- marker-start
- mathbackground
- mathcolor
- mathsize
- opacity
- paint-order
- pointer-events
- scriptlevel
- shape-rendering
- stop-color
- stop-opacity
- stroke
- stroke-dasharray
- stroke-dashoffset
- stroke-linecap
- stroke-linejoin
- stroke-miterlimit

- stroke-opacity
- stroke-width
- text-anchor
- text-decoration
- text-overflow
- text-rendering
- transform
- transform-origin
- unicode-bidi
- vector-effect
- visibility
- white-space
- word-spacing
- writing-mode

comments[p1241]

false

dataAttributes[p1241]

false

javascriptURLs[p1241]

false

elements[p1240]

All HTML elements normatively categorized as Default[p1249] within their individual definitions, alongside the MathML and SVG elements listed in the table below. Attributes included in those definitions are included in each element's respective attributes[p1240] list.

The following table lists the MathML and SVG elements that are categorized as Default[p1249] in the built-in safe default configuration[p1249], represented as a list of SanitizerElementNamespaceWithAttributes[p1240] dictionaries. For each row in the table, the "Element" column corresponds to the name[p1240] member, the "Namespace" column corresponds to the namespace[p1240] member, and the "Allowed attributes" column corresponds to the attributes[p1240] member (where each listed attribute is represented as a SanitizerAttribute[p1240] in the sequence):

Element	Namespace	Allowed attributes
a	MathML	href, hreflang, type
math	MathML	
merror	MathML	
mfrac	MathML	
mi	MathML	
mmultiscripts	MathML	
mn	MathML	
mo	MathML	fence, form, largeop, lspace, maxsize, minsize, movablelimits, rspace, separator, stretchy, symmetric
mover	MathML	accent
mpadded	MathML	depth, height, lspace, voffset, width
mphantom	MathML	
mprescripts	MathML	
mroot	MathML	
mrow	MathML	
ms	MathML	
mspace	MathML	depth, height, width
msqrt	MathML	
mstyle	MathML	
msub	MathML	
msubsup	MathML	
msup	MathML	
mtable	MathML	
mtd	MathML	columnspan, rowspan
mtext	MathML	
mtr	MathML	
munder	MathML	accentunder
munderover	MathML	accent, accentunder
semantics	MathML	

Element	Namespace	Allowed attributes
a	SVG	href, hreflang, type
circle	SVG	cx, cy, pathLength, r
defs	SVG	
desc	SVG	
ellipse	SVG	cx, cy, pathLength, rx, ry
foreignObject	SVG	height, width, x, y
g	SVG	
line	SVG	pathLength, x1, x2, y1, y2
marker	SVG	markerHeight, markerUnits, markerWidth, orient, preserveAspectRatio, refX, refY, viewBox
metadata	SVG	
path	SVG	d, pathLength
polygon	SVG	pathLength, points
polyline	SVG	pathLength, points
rect	SVG	height, pathLength, rx, ry, width, x, y
svg	SVG	height, preserveAspectRatio, viewBox, width, x, y
text	SVG	dx, dy, lengthAdjust, rotate, textLength, x, y
textPath	SVG	lengthAdjust, method, path, side, spacing, startOffset, textLength
title	SVG	
tspan	SVG	dx, dy, lengthAdjust, rotate, textLength, x, y

The **built-in navigating URL attributes list** corresponds to all HTML elements marked with **navigating URL attributes** in their normative definitions, as well as the element-attribute pairs represented in the following table. For each row in the table, the element corresponds to a SanitizerElementNamespace[p1240] whose name[p1240] member is given by the "Element" column and whose namespace[p1240] member is given by the "Element namespace" column; and the attribute corresponds to a SanitizerAttributeNamespace[p1240] whose name[p1240] member is given by the "Attribute" column and whose namespace[p1240] member is given by the "Attribute namespace" column:

Element	Element namespace	Attribute	Attribute namespace
a	SVG	href	no namespace
a	SVG	href	XLink

Note

The MathML a element is not listed here because the inner sanitize steps[p1243] remove an `href` attribute that contains a `javascript:` URL[p1245] from any MathML element, whether the attribute is in no namespace or in the XLink namespace.

The **built-in animating URL attributes list** is the list of element-attribute pairs represented by the following table. For each row in the table, the element corresponds to a SanitizerElementNamespace[p1240] whose name[p1240] member is given by the "Element" column and whose namespace[p1240] member is given by the "Element namespace" column; and the attribute corresponds to a SanitizerAttributeNamespace[p1240] whose name[p1240] member is given by the "Attribute" column and whose namespace[p1240] member is null:

Element	Element namespace	Attribute
animate	SVG	attributeName
animateTransform	SVG	attributeName
set	SVG	attributeName

The **built-in non-replaceable elements list** contains elements that must not be replaced with their children, as doing so can lead to re-parsing issues or an invalid node tree. It is the following list of SanitizerElementNamespace[p1240] dictionaries, represented by the table below. For each row in the table, the "Element" column corresponds to the name[p1240] member, and the "Element namespace" column corresponds to the namespace[p1240] member:

Element	Element namespace
html	HTML
svg	SVG
math	MathML

8.6.6 Security considerations §[p12 52]

This section is non-normative.

The Sanitizer API is intended to prevent DOM-based cross-site scripting by traversing supplied HTML content and removing elements and attributes according to a configuration. By design, the `setHTML()`[p1227] and `parseHTML()`[p1227] methods remove script-capable markup regardless of the configuration supplied; if any configuration could preserve such markup through these methods, that would be a bug.

However, there are security issues that the Sanitizer API cannot prevent. The following sections describe them.

8.6.6.1 Server-side reflected and stored XSS §[p12 52]

This section is non-normative.

The Sanitizer API operates solely in the DOM and adds a capability to traverse and filter an existing `DocumentFragment`. The Sanitizer API does not address server-side reflected or stored XSS.

8.6.6.2 DOM clobbering §[p12 52]

This section is non-normative.

DOM clobbering describes an attack in which malicious HTML confuses an application by using `id`[p162] or `name` attributes such that DOM properties, such as the `children` property of an HTML element, are shadowed by malicious content.

The Sanitizer API does not protect against DOM clobbering attacks by default, but can be configured to remove `id`[p162] and `name` attributes.

8.6.6.3 XSS with script gadgets §[p12 52]

This section is non-normative.

Script gadgets are a technique in which an attacker uses existing application code from popular JavaScript libraries to cause their own code to execute. This is often done by injecting innocent-looking code or seemingly inert DOM nodes that are only parsed and interpreted by a framework which then performs the execution of JavaScript based on that input.

The Sanitizer API cannot prevent these attacks. Instead, it relies on authors to explicitly allow unknown elements in general, and additionally to explicitly allow attributes, elements, and markup commonly used for templating and framework-specific code, such as `data-*`[p171] and `slot`[p712] attributes and elements like `slot`[p712] and `template`[p705]. These restrictions are not exhaustive and authors are encouraged to examine their third party libraries for this behavior.

8.6.6.4 Mutation XSS §[p12 52]

This section is non-normative.

Mutation XSS or mXSS describes an attack that exploits cases where the parsed DOM structure is not the same after serializing and parsing again, to bypass sanitization that happens before serialization. An example for carrying out such an attack is by relying on the change of parsing behavior for foreign content or mis-nested tags.

The Sanitizer API offers only functions that turn a string into a node tree. The context is supplied implicitly by all sanitizer functions: setHTML()[p1227] uses the current element; Document.parseHTML()[p1227] creates a new document. Therefore Sanitizer API is not directly affected by mutation XSS.

If a developer were to retrieve a sanitized node tree as a string, e.g. via `innerHTML`[p1229], and to then parse it again then mutation XSS can occur. This practice is strongly discouraged. If processing or passing of HTML as a string is necessary after all, then any string is to be considered untrusted and re-sanitized when inserted into the DOM. In other words, a sanitized and then serialized HTML tree can no longer be considered sanitized. A more complete treatment of mXSS can be found in [MXSS][p1584].

8.7 Timers §p1253

The setTimeout()p1253 and setInterval()p1253 methods allow authors to schedule timer-based callbacks.

For web developers (non-normative)

***id* = self.setTimeout(*handler* [, *timeout* [, ...*arguments*]])**

Schedules a timeout to run *handler* after *timeout* milliseconds. Any *arguments* are passed straight through to the *handler*.

***id* = self.setTimeout(*code* [, *timeout*])**

Schedules a timeout to compile and run *code* after *timeout* milliseconds.

self.clearTimeout(*id*)

Cancels the timeout set with setTimeout()p1253 or setInterval()p1253 identified by *id*.

***id* = self.setInterval(*handler* [, *timeout* [, ...*arguments*]])**

Schedules a timeout to run *handler* every *timeout* milliseconds. Any *arguments* are passed straight through to the *handler*.

***id* = self.setInterval(*code* [, *timeout*])**

Schedules a timeout to compile and run *code* every *timeout* milliseconds.

self.clearInterval(*id*)

Cancels the timeout set with setInterval()p1253 or setTimeout()p1253 identified by *id*.

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 WindowOrWorkerGlobalScopep1218 mixin have a **map of setTimeout and setInterval IDs**, which is an ordered map, initially empty. Each key in this map is a positive integer, corresponding to the return value of a setTimeout()p1253 or setInterval()p1253 call. Each value is a unique internal valuep100, corresponding to a key in the object's map of active timersp1256.

The **setTimeout(*handler*, *timeout*, ...*arguments*)** method steps are to return the result of running the timer initialization stepsp1253 given this, *handler*, *timeout*, *arguments*, and false.

The **setInterval(*handler*, *timeout*, ...*arguments*)** method steps are to return the result of running the timer initialization stepsp1253 given this, *handler*, *timeout*, *arguments*, and true.

The **clearTimeout(*id*)** and **clearInterval(*id*)** method steps are to remove this's map of setTimeout and setInterval IDsp1253[*id*].

Note

Because clearTimeout()p1253 and clearInterval()p1253 clear entries from the same map, either method can be used to clear timers created by setTimeout()p1253 or setInterval()p1253.

To perform the **timer initialization steps**, given a WindowOrWorkerGlobalScopep1218 *global*, a string or Function or TrustedScript *handler*, a number *timeout*, a list *arguments*, a boolean *repeat*, and optionally (and only if *repeat* is true) a number *previousId*, perform the following steps. They return a number.

1. If *handler* is not a Function and *previousId* was not given:
    1. Let *globalName* be "Window" if *global* is a Windowp965 object; "WorkerGlobalScope" otherwise.
    2. Let *methodName* be "setInterval" if *repeat* is true; "setTimeout" otherwise.
    3. Let *sink* be a concatenation of *globalName*, U+0020 SPACE, and *methodName*.
    4. Set *handler* to the result of invoking the get trusted type compliant string algorithm with TrustedScript, *global*, *handler*, *sink*, and "script".
2. Let *thisArg* be *global* if that is a WorkerGlobalScopep1322 object; otherwise let *thisArg* be the WindowProxyp977 that

corresponds to *global*.

3. If *previousId* was given, let *id* be *previousId*; otherwise, let *id* be an implementation-defined integer that is greater than zero and does not already exist in *global*'s map of setTimeout and setInterval IDs[p1253].

4. If the surrounding agent's event loop[p1193]'s currently running task[p1194] is a task that was created by this algorithm, then let *nestingLevel* be the task[p1194]'s timer nesting level[p1255]. Otherwise, let *nestingLevel* be 0.

   Note

   The task's timer nesting level[p1255] is used both for nested calls to `setTimeout()`[p1253], and for the repeating timers created by `setInterval()`[p1253]. (Or, indeed, for any combination of the two.) In other words, it represents nested invocations of this algorithm, not of a particular method.

5. If *timeout* is less than 0, then set *timeout* to 0.

6. If *nestingLevel* is greater than 5, and *timeout* is less than 4, then set *timeout* to 4.

7. Let *realm* be *global*'s relevant realm[p1151].

8. Let *initiating script* be the active script[p1154].

9. Let *uniqueHandle* be null.

10. Let *task* be a task[p1194] that runs the following substeps:

    1. Assert: *uniqueHandle* is a unique internal value[p100], not null.

    2. If *id* does not exist in *global*'s map of setTimeout and setInterval IDs[p1253], then abort these steps.

    3. If *global*'s map of setTimeout and setInterval IDs[p1253][*id*] does not equal *uniqueHandle*, then abort these steps.

       Note

       This accommodates for the ID having been cleared by a `clearTimeout()`[p1253] or `clearInterval()`[p1253] call, and being reused by a subsequent `setTimeout()`[p1253] or `setInterval()`[p1253] call.

    4. Let *settings object* be *global*'s relevant settings object[p1151].

    5. If scripting is disabled[p1152] for *settings object*, then abort these steps.

    6. Record timing info for timer handler given *handler*, *settings object*, and *repeat*.

    7. If *handler* is a `Function`, then invoke *handler* given *arguments* and "`report`", and with *callback this value* set to *thisArg*.

    8. Otherwise:

       1. Assert: *handler* is a string.

       2. Perform EnsureCSPDoesNotBlockStringCompilation(*realm*, « », *handler*, *handler*, timer, « », *handler*). If this throws an exception, catch it, report[p1167] it for *global*, and abort these steps.

       3. Let *fetch options* be the default script fetch options[p1154].

       4. Let *base URL* be *settings object*'s API base URL[p1144].

       5. If *initiating script* is not null:

          1. Set *fetch options* to a script fetch options[p1154] whose cryptographic nonce[p1154] is *initiating script*'s fetch options[p1153]'s cryptographic nonce[p1154], integrity metadata[p1154] is the empty string, parser metadata[p1154] is "`not-parser-inserted`", credentials mode[p1154] is *initiating script*'s fetch options[p1153]'s credentials mode[p1154], referrer policy[p1154] is *initiating script*'s fetch options[p1153]'s referrer policy[p1154], and fetch priority[p1154] is "`auto`".

          2. Set *base URL* to *initiating script*'s base URL[p1153].

       Note

       The effect of these steps ensures that the string compilation done by `setTimeout()`[p1253] and `setInterval()`[p1253] behaves equivalently to that done by `eval()`. That is, module script[p1153] fetches

via `import()` will behave the same in both contexts.

6. Let *script* be the result of creating a classic script[p1161] given *handler*, *settings object*, *base URL*, and *fetch options*.

7. Run the classic script[p1165] *script*.

9. If *id* does not exist in *global*'s map of setTimeout and setInterval IDs[p1253], then abort these steps.

10. If *global*'s map of setTimeout and setInterval IDs[p1253][*id*] does not equal *uniqueHandle*, then abort these steps.

Note

The ID might have been removed via the author code in handler *calling* `clearTimeout()`[p1253] *or* `clearInterval()`[p1253]*. Checking that uniqueHandle isn't different accounts for the possibility of the ID, after having been cleared, being reused by a subsequent* `setTimeout()`[p1253] *or* `setInterval()`[p1253] *call.*

11. If *repeat* is true, then perform the timer initialization steps[p1253] again, given *global*, *handler*, *timeout*, *arguments*, true, and *id*.

12. Otherwise, remove *global*'s map of setTimeout and setInterval IDs[p1253][*id*].

11. Increment *nestingLevel* by one.

12. Set *task*'s **timer nesting level** to *nestingLevel*.

13. Let *completionStep* be an algorithm step which queues a global task[p1195] on the **timer task source** given *global* to run *task*.

14. Set *uniqueHandle* to the result of running steps after a timeout[p1256] given *global*, "`setTimeout/setInterval`", *timeout*, and *completionStep*.

15. Set *global*'s map of setTimeout and setInterval IDs[p1253][*id*] to *uniqueHandle*.

16. Return *id*.

Note

Argument conversion as defined by Web IDL (for example, invoking `toString()` *methods on objects passed as the first argument) happens in the algorithms defined in Web IDL, before this algorithm is invoked.*

Example

So for example, the following rather silly code will result in the log containing "`ONE TWO `":

```
var log = '';
function logger(s) { log += s + ' '; }

setTimeout({ toString: function () {
  setTimeout("logger('ONE')", 100);
  return "logger('TWO')";
} }, 100);
```

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:

```
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 id = setTimeout(rescheduleWork, 0); // preschedule next iteration
  if (doExpensiveWork())
    clearTimeout(id); // clear the timeout if we don't need it
}

function scheduleWork() {
  setTimeout(rescheduleWork, 0);
}

scheduleWork(); // queues a task to do lots of work
```

Objects that implement the WindowOrWorkerGlobalScope[p1218] mixin have a **map of active timers**, which is an ordered map, initially empty. Each key in this map is a unique internal value[p100] that represents a timer, and each value is a DOMHighResTimeStamp, representing the expiry time for that timer.

To **run steps after a timeout**, given a WindowOrWorkerGlobalScope[p1218] *global*, a string *orderingIdentifier*, a number *milliseconds*, and a set of steps *completionSteps*, perform the following steps. They return a unique internal value[p100].

1. Let *timerKey* be a new unique internal value[p100].
2. Let *startTime* be the current high resolution time given *global*.
3. Set *global*'s map of active timers[p1256][*timerKey*] to *startTime* plus *milliseconds*.
4. Run the following steps in parallel[p44]:
    1. If *global* is a Window[p965] object, wait until *global*'s associated Document[p966] has been fully active[p1051] for a further *milliseconds* milliseconds (not necessarily consecutively).

       Otherwise, *global* is a WorkerGlobalScope[p1322] object; wait until *milliseconds* milliseconds have passed with the worker not suspended (not necessarily consecutively).
    2. Wait until any invocations of this algorithm that had the same *global* and *orderingIdentifier*, that started before this one, and whose *milliseconds* is less than or equal to this one's, have completed.
    3. Optionally, wait a further implementation-defined length of time.

       Note

       This is intended to allow user agents to pad timeouts as needed to optimize the power usage of the device. For example, some processors have a low-power mode where the granularity of timers is reduced; on such platforms, user agents can slow timers down to fit this schedule instead of requiring the processor to use the more accurate mode with its associated higher power usage.
    4. Perform *completionSteps*.
    5. Remove *global*'s map of active timers[p1256][*timerKey*].
5. Return *timerKey*.

Note

Run steps after a timeout[p1256] *is meant to be used by other specifications that want to execute developer-supplied code after a developer-supplied timeout, in a similar manner to* setTimeout()[p1253]*. (Note, however, it does not have the nesting and clamping behavior of* setTimeout()[p1253]*.) Such specifications can choose an* orderingIdentifier *to ensure ordering within their specification's timeouts, while not constraining ordering with respect to other specification's timeouts.*

8.8 Microtask queuing §[p1257]

For web developers (non-normative)

***self*.queueMicrotask[p1257](*callback*)**

Queues[p1195] a microtask[p1194] to run the given *callback*.

The **queueMicrotask(*callback*)** method must queue a microtask[p1195] to invoke *callback* with « » and "report".

The queueMicrotask()[p1257] method allows authors to schedule a callback on the microtask queue[p1194]. 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[p1193], as would be the case when using, for example, setTimeout(*f*, 0)[p1253].

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()[p1281] 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()[p1281].

As can be seen from the following examples, the best way of thinking about queueMicrotask()[p1257] 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()[p1257] 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:

```
MyElement.prototype.loadData = function (url) {
  if (this._cache[url]) {
    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"));
    });
  }
};
```

This naïve implementation is problematic, however, in that it causes its users to experience inconsistent behavior. For example, code such as

```
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()[p1257] can be used:

```
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:

```
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()`).

8.9 User prompts

8.9.1 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[p1260] 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[p1260] *message*.
5. Let *userPromptHandler* be WebDriver BiDi user prompt opened with this, "`alert`", and *message*.
6. If *userPromptHandler* is "`none`":
    1. Show *message* to the user, treating U+000A LF as a line break.
    2. Optionally, pause[p1203] while waiting for the user to acknowledge the message.
7. Invoke WebDriver BiDi user prompt closed with this, "`alert`", and true.

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[p1260] 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[p1260] *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. Let *userPromptHandler* be WebDriver BiDi user prompt opened with this, "`confirm`", and *message*.
6. Let *accepted* be false.
7. If *userPromptHandler* is "`none`":
    1. Pause[p1203] until the user responds either positively or negatively.
    2. If the user responded positively, then set *accepted* to true.
8. If *userPromptHandler* is "`accept`", then set *accepted* to true.
9. Invoke WebDriver BiDi user prompt closed with this, "`confirm`", and *accepted*.
10. Return *accepted*.

The **`prompt(message, default)`** method steps are:

1. If we cannot show simple dialogs[p1260] 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[p1260] *message*.
4. Set *default* to the result of optionally truncating[p1260] *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. Let *userPromptHandler* be WebDriver BiDi user prompt opened with this, "`prompt`", and *message*.

7. Let *result* be null.
8. If *userPromptHandler* is "`none`":
    1. Pause[p1203] while waiting for the user's response.
    2. If the user did not abort, then set *result* to the string that the user responded with.
9. Otherwise, if *userPromptHandler* is "`accept`", then set *result* to the empty string.
10. Invoke WebDriver BiDi user prompt closed with this, "`prompt`", false if *result* is null or true otherwise, and *result*.
11. Return *result*.

To **optionally truncate a simple dialog string** *s*, return either *s* itself or some string derived from *s* that is shorter. User agents should not provide UI for displaying the elided portion of *s*, as this makes it too easy for abusers to create dialogs of the form "Important security alert! Click 'Show More' for full details!".

Note

For example, a user agent might want to only display the first 100 characters of a message. Or, a user agent might replace the middle of the string with "...". These types of modifications can be useful in limiting the abuse potential of unnaturally large, trustworthy-looking system dialogs.

We **cannot show simple dialogs** for a `Window`[p965] *window* when the following algorithm returns true:

1. If the active sandboxing flag set[p959] of *window*'s associated `Document`[p966] has the sandboxed modals flag[p958] set, then return true.
2. If *window*'s relevant agent[p1141]'s event loop[p1193]'s termination nesting level[p1114] is nonzero, then optionally return true.
3. 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.)
4. Return false.

8.9.2 Printing §[p1260]

For web developers (non-normative)

***window*.print[p1260]()**

Prompts the user to print the page.

The **`print()`** method steps are:

1. Let *document* be this's associated `Document`[p966].
2. If *document* is not fully active[p1051], then return.
3. If *document*'s unload counter[p1114] is greater than 0, then return.
4. If *document* is ready for post-load tasks[p1459], then run the printing steps[p1260] for *document*.
5. Otherwise, set *document*'s **print when loaded** flag.

User agents should also run the printing steps[p1260] whenever the user asks for the opportunity to obtain a physical form[p1528] (e.g. printed copy), or the representation of a physical form (e.g. PDF copy), of a document.

The **printing steps** for a `Document`[p135] *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()`[p1260] 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[p959] of *document* has the sandboxed modals flag[p958] set, then return.

Note

If the printing dialog is blocked by a `Document`[p135]*'s sandbox, then neither the* `beforeprint`[p1574] *nor* `afterprint`[p1574] *events will be fired.*

3. The user agent must fire an event named `beforeprint`[p1574] at the relevant global object[p1151] of *document*, as well as any child navigable[p1039] in it.

Firing in children only doesn't seem right here, and some tasks likely need to be queued. See issue #5096.

Example

The `beforeprint`[p1574] 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[p1528] (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[p1203] 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`[p1574] at the relevant global object[p1151] of *document*, as well as any child navigables[p1039] in it.

Firing in children only doesn't seem right here, and some tasks likely need to be queued. See issue #5096.

Example

The `afterprint`[p1574] 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.

8.10 System state and capabilities §[p12] 61

8.10.1 The `Navigator`[p1261] object §[p12] 61

Instances of `Navigator`[p1261] represent the identity and state of the user agent (the client). It has also been used as a generic global under which various APIs are located, but this is not precedent to build upon. Instead use the `WindowOrWorkerGlobalScope`[p1218] mixin or equivalent.

IDL

```
[Exposed=Window]
interface Navigator {
  // objects implementing this interface also implement the interfaces given below
};
Navigator includes NavigatorID;
Navigator includes NavigatorLanguage;
Navigator includes NavigatorOnLine;
Navigator includes NavigatorContentUtils;
Navigator includes NavigatorCookies;
Navigator includes NavigatorPlugins;
Navigator includes NavigatorConcurrentHardware;
```

Note

These interface mixins are defined separately so that `WorkerNavigator`[p1337] *can reuse parts of the* `Navigator`[p1261] *interface.*

Each Window[p965] has an **associated Navigator**, which is a Navigator[p1261] object. Upon creation of the Window[p965] object, its associated Navigator[p1262] must be set to a new Navigator[p1261] object created in the Window[p965] object's relevant realm[p1151].

The **navigator** and **clientInformation** getter steps are to return this's associated Navigator[p1262].

MDN

8.10.1.1 Client identification §[p1262]

IDL

```
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[p1262] constrains the NavigatorID[p1262] mixin to the combinations of attribute values and presence of taintEnabled()[p1263] and oscpu[p1263] 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[p1262].appCodeName[p1263]**
Returns the string "Mozilla".

***self*.navigator[p1262].appName[p1263]**
Returns the string "Netscape".

***self*.navigator[p1262].appVersion[p1263]**
Returns the version of the browser.

***self*.navigator[p1262].platform[p1263]**
Returns the name of the platform.

***self*.navigator[p1262].product[p1263]**
Returns the string "Gecko".

***window*.navigator[p1262].productSub[p1263]**
Returns either the string "20030107", or the string "20100101".

***self*.navigator[p1262].userAgent[p1263]**
Returns the complete `User-Agent` header.

***window*.navigator[p1262].vendor[p1263]**
Returns either the empty string, the string "Apple Computer, Inc.", or the string "Google Inc.".

***window*.navigator[p1262].vendorSub[p1263]**
Returns the empty string.

The **appCodeName** getter steps are to return "Mozilla".

The **appName** getter steps are to return "Netscape".

The **appVersion** getter steps are:

1. Let *userAgent* be this's relevant settings object[p1151]'s environment default `User-Agent` value.
2. If *userAgent* does not start with `Mozilla/5.0 (`, then return the empty string.
3. Let *trail* be the substring of *userAgent*, isomorphic decoded, that follows the "Mozilla/" prefix.
4. ↪ **If the navigator compatibility mode[p1262] is *Chrome* or *WebKit***
   Return *trail*.

   ↪ **If the navigator compatibility mode[p1262] is *Gecko***
   If *trail* starts with "5.0 (Windows", then return "5.0 (Windows)".

   Otherwise, return the prefix of *trail* up to but not including the first U+003B (;), concatenated with the character U+0029 RIGHT PARENTHESIS. For example, "5.0 (Macintosh)", "5.0 (Android 10)", or "5.0 (X11)".

The **platform** getter steps are to return a string representing the platform on which the browser is executing (e.g. "MacIntel", "Win32", "Linux x86_64", "Linux armv81") or, for privacy and compatibility, a string that is commonly returned on another platform.

The **product** getter steps are to return "Gecko".

The **productSub** getter steps are to return the appropriate string from the following list:

↪ **If the navigator compatibility mode[p1262] is *Chrome* or *WebKit***
The string "20030107".

↪ **If the navigator compatibility mode[p1262] is *Gecko***
The string "20100101".

The **userAgent** getter steps are to return this's relevant settings object[p1151]'s environment default `User-Agent` value.

The **vendor** getter steps are to return the appropriate string from the following list:

↪ **If the navigator compatibility mode[p1262] is *Chrome***
The string "Google Inc.".

↪ **If the navigator compatibility mode[p1262] is *Gecko***
The empty string.

↪ **If the navigator compatibility mode[p1262] is *WebKit***
The string "Apple Computer, Inc.".

The **vendorSub** getter steps are to return the empty string.

If the navigator compatibility mode[p1262] 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.

8.10.1.2 Language preferences §[p1264]

IDL

```
interface mixin NavigatorLanguage {
  readonly attribute DOMString language;
  readonly attribute FrozenArray<DOMString> languages;
};
```

For web developers (non-normative)

***self*.navigator[p1262].language[p1264]**

Returns a language tag representing the user's preferred language.

***self*.navigator[p1262].languages[p1264]**

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[p1264].

Note

A languagechange[p1575] *event is fired at the* Window[p965] *or* WorkerGlobalScope[p1322] *object when the user agent's understanding of what the user's preferred languages are changes.*

The **language** getter steps are:

1. Let *emulatedLanguage* be the WebDriver BiDi emulated language for this's relevant settings object[p1151].
2. If *emulatedLanguage* is not null, return *emulatedLanguage*.
3. Return a valid BCP 47 language tag representing either a plausible language[p1264] or the user's most preferred language. [BCP47][p1579]

The **languages** getter steps are:

1. Let *emulatedLanguage* be the WebDriver BiDi emulated language for this's relevant settings object[p1151].
2. If *emulatedLanguage* is not null, return a frozen array containing *emulatedLanguage*.
3. Return a frozen array of valid BCP 47 language tags representing either one or more plausible languages[p1264], or the user's preferred languages, ordered by preference with the most preferred language first. [BCP47][p1579]

The same object must be returned until the user agent needs to return different values, or values in a different order, or *emulatedLanguage* is updated.

Whenever the user agent needs to make the navigator.languages[p1264] attribute of a Window[p965] or WorkerGlobalScope[p1322] object *global* return a new set of language tags, the user agent must queue a global task[p1195] on the DOM manipulation task source[p1203] given *global* to fire an event named languagechange[p1575] 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][p1587]
- 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.

8.10.1.3 Browser state §[p1265]

IDL
```
interface mixin NavigatorOnLine {
  readonly attribute boolean onLine;
};
```

For web developers (non-normative)

***self*.navigator[p1262].onLine[p1265]**

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[p1576] and offline[p1576] are fired when the value of this attribute changes.

The **onLine** getter steps are to return true if this's relevant settings object[p1151] is offline is false; otherwise false.

When the value that would be returned by the navigator.onLine[p1265] attribute of a Window[p965] or WorkerGlobalScope[p1322] *global* changes from true to false, the user agent must queue a global task[p1195] on the networking task source[p1204] given *global* to fire an event named offline[p1576] at *global*.

On the other hand, when the value that would be returned by the navigator.onLine[p1265] attribute of a Window[p965] or WorkerGlobalScope[p1322] *global* changes from false to true, the user agent must queue a global task[p1195] on the networking task source[p1204] given *global* to fire an event named online[p1576] at the Window[p965] or WorkerGlobalScope[p1322] object.

Note

This attribute is inherently unreliable. A computer can be connected to a network without having Internet access.

Example

In this example, an indicator is updated as the browser goes online and offline.

```
<!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>
 </body>
</html>
```

MDN

8.10.1.4 Custom scheme handlers: the registerProtocolHandler()[p1266] method §[p1265]

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[p1262].registerProtocolHandler[p1266](*scheme*, *url*)**

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][p1586]

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[p1262].unregisterProtocolHandler[p1267](*scheme*, *url*)**

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[p1267] with *scheme*, *url*, and this's relevant settings object[p1151].

2. In parallel[p44]: **register a protocol 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.

   If the `registerProtocolHandler()` automation mode[p1268] of this's relevant global object[p1151]'s associated `Document`[p966] is not "**`none`**", the user agent should first verify that it is in an automation context (see WebDriver's security considerations). The user agent should then bypass the above communication of information and gathering of user consent, and instead do the following based on the value of the `registerProtocolHandler()` automation mode[p1268]:

   "`autoAccept`"
   Act as if the user has seen the registration details and accepted the request.

   "`autoReject`"
   Act as if the user has seen the registration details and rejected the request.

   When the **user agent uses this handler** for a URL *inputURL*:

   1. Assert: *inputURL*'s scheme is *normalizedScheme*.
   2. Set the username given *inputURL* and the empty string.
   3. Set the password given *inputURL* and the empty string.
   4. Let *inputURLString* be the serialization of *inputURL*.
   5. Let *encodedURL* be the result of running UTF-8 percent-encode on *inputURLString* using the component percent-encode set.
   6. Let *handlerURLString* be *normalizedURLString*.
   7. Replace the first instance of "`%s`" in *handlerURLString* with *encodedURL*.
   8. Let *resultURL* be the result of parsing *handlerURLString*.
   9. Navigate[p1063] an appropriate navigable[p1035] to *resultURL*.

Example

If the user had visited a site at `https://example.com/` that made the following call:

```
navigator.registerProtocolHandler('web+soup', 'soup?url=%s')
```

...and then, much later, while visiting `https://www.example.net/`, clicked on a link such as:

```
<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[p1063] 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[p1267] with *scheme*, *url*, and this's relevant settings object[p1151].
2. In parallel[p44]: 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[p1144] *environment*, run these steps:

1. Set *scheme* to *scheme*, converted to ASCII lowercase.
2. If *scheme* is neither a safelisted scheme[p1267] 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`
   - `ftp`
   - `ftps`
   - `geo`
   - `im`
   - `irc`
   - `ircs`
   - `magnet`
   - `mailto`
   - `matrix`
   - `mms`
   - `news`
   - `nntp`
   - `openpgp4fpr`
   - `sftp`
   - `sip`
   - `sms`
   - `smsto`
   - `ssh`
   - `tel`
   - `urn`
   - `webcal`
   - `wtai`
   - `xmpp`

   Note

   This list can be changed. If there are schemes that ought to be added, please send feedback.
3. If *url* does not contain "`%s`", then throw a "`SyntaxError`" `DOMException`.
4. Let *urlRecord* be the result of encoding-parsing a URL[p101] given *url*, relative to *environment*.

5. If *urlRecord* is failure, then throw a "SyntaxError" DOMException.

   Note

   This is forcibly the case if the `%s` placeholder is in the host or port of the URL.

6. If *urlRecord*'s scheme is not an HTTP(S) scheme or *urlRecord*'s origin is not same origin[p940] with *environment*'s origin[p1144], then throw a "SecurityError" DOMException.

7. Assert: the result of Is url potentially trustworthy? given *urlRecord* is "`Potentially Trustworthy`".

   Note

   Because normalize protocol handler parameters[p1267] is run within a secure context[p1152], this is implied by the same origin[p940] condition.

8. Return (*scheme*, *urlRecord*).

   Note

   The serialization of urlRecord *will by definition not be a valid URL string as it includes the string "`%s`" which is not a valid component in a URL.*

8.10.1.4.1 Security and privacy §[p1268]

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. Implementers might wish to consider allowing administrators to disable custom handlers on certain subdomains, content types, or schemes.

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.

8.10.1.4.2 User agent automation §[p1268]

Each Document[p135] has a **`registerProtocolHandler()` automation mode**. It defaults to "none[p1266]", but it also can be either "autoAccept[p1266]" or "autoReject[p1266]".

For the purposes of user agent automation and website testing, this standard defines **Set RPH Registration Mode** WebDriver extension command. It instructs the user agent to place a Document[p135] into a mode where it will automatically simulate a user either accepting or rejecting and registration confirmation prompt dialog.

HTTP Method	URI Template
`POST`	/session/{session id}/custom-handlers/set-mode

The remote end steps are:

1. If *parameters* is not a JSON Object, return a WebDriver error with WebDriver error code invalid argument.
2. Let *mode* be the result of getting a property named "mode" from *parameters*.
3. If *mode* is not "autoAccept[p1266]", "autoReject[p1266]", or "none[p1266]", return a WebDriver error with WebDriver error code invalid argument.
4. Let *document* be the current browsing context's active document[p1046].
5. Set *document*'s registerProtocolHandler() automation mode[p1268] to *mode*.
6. Return success with data null.

8.10.1.5 Cookies §[p12 69]

IDL

```
interface mixin NavigatorCookies {
  readonly attribute boolean cookieEnabled;
};
```

For web developers (non-normative)

***window*.navigator[p1262].cookieEnabled[p1269]**

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][p1580]

8.10.1.6 PDF viewing support §[p12 69]

For web developers (non-normative)

***window*.navigator[p1262].pdfViewerEnabled[p1271]**

Returns true if the user agent supports inline viewing of PDF files when navigating[p1063] to them, or false otherwise. In the latter case, PDF files will be handled by external software[p1073].

IDL

```
interface mixin NavigatorPlugins {
  [SameObject] readonly attribute PluginArray plugins;
  [SameObject] readonly attribute MimeTypeArray mimeTypes;
  boolean javaEnabled();
  readonly attribute boolean pdfViewerEnabled;
};

[Exposed=Window,
 LegacyUnenumerableNamedProperties]
interface PluginArray {
  undefined refresh();
  readonly attribute unsigned long length;
  getter Plugin? item(unsigned long index);
  getter Plugin? namedItem(DOMString name);
};

[Exposed=Window,
 LegacyUnenumerableNamedProperties]
interface MimeTypeArray {
```

```
  readonly attribute unsigned long length;
  getter MimeType? item(unsigned long index);
  getter MimeType? namedItem(DOMString name);
};

[Exposed=Window,
 LegacyUnenumerableNamedProperties]
interface Plugin {
  readonly attribute DOMString name;
  readonly attribute DOMString description;
  readonly attribute DOMString filename;
  readonly attribute unsigned long length;
  getter MimeType? item(unsigned long index);
  getter MimeType? namedItem(DOMString name);
};

[Exposed=Window]
interface MimeType {
  readonly attribute DOMString type;
  readonly attribute DOMString description;
  readonly attribute DOMString suffixes;
  readonly attribute Plugin enabledPlugin;
};
```

Although these days detecting PDF viewer support can be done via navigator.pdfViewerEnabled[p1271], for historical reasons, there are a number of complex and intertwined interfaces that provide the same capability, which legacy code relies on. This section specifies both the simple modern variant and the complicated historical one.

Each user agent has a **PDF viewer supported** boolean, whose value is implementation-defined (and might vary according to user preferences).

Note

This value also impacts the navigation[p1063] processing model.

Each Window[p965] object has a **PDF viewer plugin objects** list. If the user agent's PDF viewer supported[p1270] is false, then it is the empty list. Otherwise, it is a list containing five Plugin[p1270] objects, whose names[p1271] are, respectively:

0. "PDF Viewer"
1. "Chrome PDF Viewer"
2. "Chromium PDF Viewer"
3. "Microsoft Edge PDF Viewer"
4. "WebKit built-in PDF"

The values of the above list form the **PDF viewer plugin names** list.

Note

These names were chosen based on evidence of what websites historically search for, and thus what is necessary for user agents to expose in order to maintain compatibility with existing content. They are ordered alphabetically. The "PDF Viewer" name was then inserted in the 0th position so that the enabledPlugin[p1272] getter could point to a generic plugin name.

Each Window[p965] object has a **PDF viewer mime type objects** list. If the user agent's PDF viewer supported[p1270] is false, then it is the empty list. Otherwise, it is a list containing two MimeType[p1270] objects, whose types[p1272] are, respectively:

0. "application/pdf"
1. "text/pdf"

The values of the above list form the **PDF viewer mime types** list.

Each NavigatorPlugins[p1269] object has a **plugins array**, which is a new PluginArray[p1269], and a **mime types array**, which is a new MimeTypeArray[p1269].

The NavigatorPlugins[p1269] mixin's **plugins** getter steps are to return this's plugins array[p1270].

The NavigatorPlugins[p1269] mixin's **mimeTypes** getter steps are to return this's mime types array[p1270].

The NavigatorPlugins[p1269] mixin's **javaEnabled()** method steps are to return false.

The NavigatorPlugins[p1269] mixin's **pdfViewerEnabled** getter steps are to return the user agent's PDF viewer supported[p1270].

The PluginArray[p1269] interface supports named properties. If the user agent's PDF viewer supported[p1270] is true, then they are the PDF viewer plugin names[p1270]. Otherwise, they are the empty list.

The PluginArray[p1269] interface's **namedItem(*name*)** method steps are:

1. For each Plugin[p1270] *plugin* of this's relevant global object[p1151]'s PDF viewer plugin objects[p1270]: if *plugin*'s name[p1271] is *name*, then return *plugin*.
2. Return null.

The PluginArray[p1269] interface supports indexed properties. The supported property indices are the indices of this's relevant global object[p1151]'s PDF viewer plugin objects[p1270].

The PluginArray[p1269] interface's **item(*index*)** method steps are:

1. Let *plugins* be this's relevant global object[p1151]'s PDF viewer plugin objects[p1270].
2. If *index* < *plugins*'s size, then return *plugins*[*index*].
3. Return null.

The PluginArray[p1269] interface's **length** getter steps are to return this's relevant global object[p1151]'s PDF viewer plugin objects[p1270]'s size.

The PluginArray[p1269] interface's **refresh()** method steps are to do nothing.

The MimeTypeArray[p1269] interface supports named properties. If the user agent's PDF viewer supported[p1270] is true, then they are the PDF viewer mime types[p1270]. Otherwise, they are the empty list.

The MimeTypeArray[p1269] interface's **namedItem(*name*)** method steps are:

1. For each MimeType[p1270] *mimeType* of this's relevant global object[p1151]'s PDF viewer mime type objects[p1270]: if *mimeType*'s type[p1272] is *name*, then return *mimeType*.
2. Return null.

The MimeTypeArray[p1269] interface supports indexed properties. The supported property indices are the indices of this's relevant global object[p1151]'s PDF viewer mime type objects[p1270].

The MimeTypeArray[p1269] interface's **item(*index*)** method steps are:

1. Let *mimeTypes* be this's relevant global object[p1151]'s PDF viewer mime type objects[p1270].
2. If *index* < *mimeTypes*'s size, then return *mimeTypes*[*index*].
3. Return null.

The MimeTypeArray[p1269] interface's **length** getter steps are to return this's relevant global object[p1151]'s PDF viewer mime type objects[p1270]'s size.

Each Plugin[p1270] object has a **name**, which is set when the object is created.

The Plugin[p1270] interface's **name** getter steps are to return this's name[p1271].

The Plugin[p1270] interface's **description** getter steps are to return "`Portable Document Format`".

The Plugin[p1270] interface's **filename** getter steps are to return "internal-pdf-viewer".

The Plugin[p1270] interface supports named properties. If the user agent's PDF viewer supported[p1270] is true, then they are the PDF viewer mime types[p1270]. Otherwise, they are the empty list.

The Plugin[p1270] interface's **namedItem(*name*)** method steps are:

1. For each MimeType[p1270] *mimeType* of this's relevant global object[p1151]'s PDF viewer mime type objects[p1270]: if *mimeType*'s type[p1272] is *name*, then return *mimeType*.
2. Return null.

The Plugin[p1270] interface supports indexed properties. The supported property indices are the indices of this's relevant global object[p1151]'s PDF viewer mime type objects[p1270].

The Plugin[p1270] interface's **item(*index*)** method steps are:

1. Let *mimeTypes* be this's relevant global object[p1151]'s PDF viewer mime type objects[p1270].
2. If *index* < *mimeTypes*'s size, then return *mimeTypes*[*index*].
3. Return null.

The Plugin[p1270] interface's **length** getter steps are to return this's relevant global object[p1151]'s PDF viewer mime type objects[p1270]'s size.

Each MimeType[p1270] object has a **type**, which is set when the object is created.

The MimeType[p1270] interface's **type** getter steps are to return this's type[p1272].

The MimeType[p1270] interface's **description** getter steps are to return "Portable Document Format".

The MimeType[p1270] interface's **suffixes** getter steps are to return "pdf".

The MimeType[p1270] interface's **enabledPlugin** getter steps are to return this's relevant global object[p1151]'s PDF viewer plugin objects[p1270][0] (i.e., the generic "PDF Viewer" one).

8.11 Images §[p1272]

8.11.1 The ImageData[p1272] interface §[p1272]

IDL

```
typedef (Uint8ClampedArray or Float16Array) ImageDataArray;

enum ImageDataPixelFormat { "rgba-unorm8", "rgba-float16" };

dictionary ImageDataSettings {
  PredefinedColorSpace colorSpace;
  ImageDataPixelFormat pixelFormat = "rgba-unorm8";
};

[Exposed=(Window,Worker),
 Serializable]
interface ImageData {
  constructor(unsigned long sw, unsigned long sh, optional ImageDataSettings settings = {});
  constructor(ImageDataArray data, unsigned long sw, optional unsigned long sh, optional
ImageDataSettings settings = {});

  readonly attribute unsigned long width;
  readonly attribute unsigned long height;
```

```
  readonly attribute ImageDataArray data;
  readonly attribute ImageDataPixelFormat pixelFormat;
  readonly attribute PredefinedColorSpace colorSpace;
};
```

An ImageData[p1272] object **represents a rectanglar bitmap** with width equal to the width[p1274] attribute and height equal to the height[p1274] attribute. The pixel values of this bitmap are stored in the data[p1274] attribute in left-to-right order, row by row from top to bottom, starting with 0 for the top left pixel, with the order and numerical representation of the color components of each pixel determined by the pixelFormat[p1274] attribute. The color space of the pixel values of the bitmap is determined by the colorSpace[p1274] attribute.

For web developers (non-normative)

***imageData* = new ImageData[p1273](*sw*, *sh* [, *settings*])**

Returns an ImageData[p1272] object with the given dimensions and the color space indicated by *settings*. All the pixels in the returned object are transparent black.

Throws an "IndexSizeError" DOMException if either of the width or height arguments are zero.

***imageData* = new ImageData[p1273](*data*, *sw* [, *sh* [, *settings*]])**

Returns an ImageData[p1272] object using the data provided in the ImageDataArray[p1272] argument, interpreted using the given dimensions and the color space indicated by *settings*.

The byte length of the data needs to be a multiple of the number of bytes per pixel times the given width. If the height is provided as well, then the length needs to be exactly the number of bytes per pixel times the width times the height.

Throws an "IndexSizeError" DOMException if the given data and dimensions can't be interpreted consistently, or if either dimension is zero.

***imageData*.width[p1274]**

***imageData*.height[p1274]**

Returns the actual dimensions of the data in the ImageData[p1272] object, in pixels.

***imageData*.data[p1274]**

Returns the one-dimensional array containing the data in RGBA order, as integers in the range 0 to 255.

***imageData*.colorSpace[p1274]**

Returns the color space of the pixels.

The **new ImageData(*sw*, *sh*, *settings*)** constructor steps are:

1. If one or both of *sw* and *sh* are zero, then throw an "IndexSizeError" DOMException.
2. Initialize[p1274] this given *sw*, *sh*, and *settings*.
3. Initialize the image data of this to transparent black.

The **new ImageData(*data*, *sw*, *sh*, *settings*)** constructor steps are:

1. Let *bytesPerPixel* be 4 if *settings*["pixelFormat[p1274]"] is "rgba-unorm8[p1275]"; otherwise 8.
2. Let *length* be the buffer source byte length of *data*.
3. If *length* is not a nonzero integral multiple of *bytesPerPixel*, then throw an "InvalidStateError" DOMException.
4. Let *length* be *length* divided by *bytesPerPixel*.
5. 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.*
6. Let *height* be *length* divided by *sw*.
7. If *sh* was given and its value is not equal to *height*, then throw an "IndexSizeError" DOMException.
8. Initialize[p1274] this given *sw*, *sh*, *settings*, and *source*[p1274] set to *data*.

Note

This step does not set this's data to a copy of data. *It sets it to the actual* ImageDataArray[p1272] *object passed as* data.

To **initialize an ImageData object** *imageData*, given a positive integer number of pixels per row *pixelsPerRow*, a positive integer number of rows *rows*, an ImageDataSettings[p1272] *settings*, an optional ImageDataArray[p1272] ***source***, and an optional PredefinedColorSpace[p782] ***defaultColorSpace***:

1. If *source* was given:
    1. If *settings*["pixelFormat[p1274]"] equals "rgba-unorm8[p1275]" and *source* is not a Uint8ClampedArray, then throw an "InvalidStateError" DOMException.
    2. If *settings*["pixelFormat[p1274]"] is "rgba-float16[p1275]" and *source* is not a Float16Array, then throw an "InvalidStateError" DOMException.
    3. Initialize the **data** attribute of *imageData* to *source*.
2. Otherwise (*source* was not given):
    1. If *settings*["pixelFormat[p1274]"] is "rgba-unorm8[p1275]", then initialize the data[p1274] attribute of *imageData* to a new Uint8ClampedArray object. The Uint8ClampedArray object must use a new ArrayBuffer for its storage, and must have a zero byte offset and byte length equal to the length of its storage, in bytes. The storage ArrayBuffer must have a length of 4 × *rows* × *pixelsPerRow* bytes.
    2. Otherwise, if *settings*["pixelFormat[p1274]"] is "rgba-float16[p1275]", then initialize the data[p1274] attribute of *imageData* to a new Float16Array object. The Float16Array object must use a new ArrayBuffer for its storage, and must have a zero byte offset and byte length equal to the length of its storage, in bytes. The storage ArrayBuffer must have a length of 8 × *rows* × *pixelsPerRow* bytes.
    3. If the storage ArrayBuffer could not be allocated, then rethrow the RangeError thrown by JavaScript, and return.
3. Initialize the **width** attribute of *imageData* to *pixelsPerRow*.
4. Initialize the **height** attribute of *imageData* to *rows*.
5. Initialize the **pixelFormat** attribute of *imageData* to *settings*["**pixelFormat**"].
6. If *settings*["colorSpace[p1274]"] exists, then initialize the **colorSpace** attribute of *imageData* to *settings*["**colorSpace**"].
7. Otherwise, if *defaultColorSpace* was given, then initialize the colorSpace[p1274] attribute of *imageData* to *defaultColorSpace*.
8. Otherwise, initialize the colorSpace[p1274] attribute of *imageData* to "srgb[p782]".

ImageData[p1272] objects are serializable objects[p122]. Their serialization steps[p122], given *value* and *serialized*, are:

1. Set *serialized*.[[Data]] to the sub-serialization[p127] of the value of *value*'s data[p1274] attribute.
2. Set *serialized*.[[Width]] to the value of *value*'s width[p1274] attribute.
3. Set *serialized*.[[Height]] to the value of *value*'s height[p1274] attribute.
4. Set *serialized*.[[ColorSpace]] to the value of *value*'s colorSpace[p1274] attribute.
5. Set *serialized*.[[PixelFormat]] to the value of *value*'s pixelFormat[p1274] attribute.

Their deserialization steps[p122], given *serialized*, *value*, and *targetRealm*, are:

1. Initialize *value*'s data[p1274] attribute to the sub-deserialization[p130] of *serialized*.[[Data]].
2. Initialize *value*'s width[p1274] attribute to *serialized*.[[Width]].
3. Initialize *value*'s height[p1274] attribute to *serialized*.[[Height]].
4. Initialize *value*'s colorSpace[p1274] attribute to *serialized*.[[ColorSpace]].
5. Initialize *value*'s pixelFormat[p1274] attribute to *serialized*.[[PixelFormat]].

The ImageDataPixelFormat[p1272] enumeration is used to specify type of the data[p1274] attribute of an ImageData[p1272] and the arrangement and numerical representation of the color components for each pixel.

The "**rgba-unorm8**" value indicates that the data[p1274] attribute of an ImageData[p1272] must be of type Uint8ClampedArray. The color components of each pixel must be stored in four sequential elements in the order of red, green, blue, and then alpha. Each element represents the 8-bit unsigned normalized value for that component.

The "**rgba-float16**" value indicates that the data[p1274] attribute of an ImageData[p1272] must be of type Float16Array. The color components of each pixel must be stored in four sequential elements in the order of red, green, blue, and then alpha. Each element represents the value for that component.

8.11.2 The ImageBitmap[p1275] interface §[p1275]

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 { "from-image", "flipY" };
enum PremultiplyAlpha { "none", "premultiply", "default" };
enum ColorSpaceConversion { "none", "default" };
enum ResizeQuality { "pixelated", "low", "medium", "high" };

dictionary ImageBitmapOptions {
  ImageOrientation imageOrientation = "from-image";
  PremultiplyAlpha premultiplyAlpha = "default";
  ColorSpaceConversion colorSpaceConversion = "default";
  [EnforceRange] unsigned long resizeWidth;
  [EnforceRange] unsigned long resizeHeight;
  ResizeQuality resizeQuality = "low";
};
```

An ImageBitmap[p1275] object represents a bitmap image that can be painted to a canvas without undue latency.

Note

The exact judgement of what is undue latency of this is left up to the implementer, but in general if making use of the bitmap requires network I/O, or even local disk I/O, then the latency is probably undue; whereas if it only requires a blocking read from a GPU or system RAM, the latency is probably acceptable.

For web developers (non-normative)

***promise* = self.createImageBitmap[p1276](*image* [, *options*])**

***promise* = self.createImageBitmap[p1276](*image*, *sx*, *sy*, *sw*, *sh* [, *options*])**

Takes *image*, which can be an img[p359] element, an SVG image element, a video[p420] element, a canvas[p713] element, a Blob object, an ImageData[p1272] object, or another ImageBitmap[p1275] object, and returns a promise that is resolved when a new ImageBitmap[p1275] is created.

If no ImageBitmap[p1275] object can be constructed, for example because the provided *image* data is not actually an image, then the promise is rejected instead.

If *sx*, *sy*, *sw*, and *sh* arguments are provided, the source image is cropped to the given pixels, with any pixels missing in the original replaced by transparent black. These coordinates are in the source image's pixel coordinate space, *not* in CSS pixels.

If *options* is provided, the ImageBitmap[p1275] object's bitmap data is modified according to *options*. For example, if the premultiplyAlpha[p1279] option is set to "premultiply[p1279]", the bitmap data[p1276]'s non-alpha color components are premultiplied by the alpha component[p784].

Rejects the promise with an "InvalidStateError" DOMException if the source image is not in a valid state (e.g., an img[p359]

element that hasn't loaded successfully, an ImageBitmap[p1275] object whose [[Detached]][p124] internal slot value is true, an ImageData[p1272] object whose data[p1274] attribute value's [[ViewedArrayBuffer]] internal slot is detached, or a Blob whose data cannot be interpreted as a bitmap image).

***imageBitmap*.close[p1279]()**

Releases *imageBitmap*'s underlying bitmap data[p1276].

***imageBitmap*.width[p1279]**

Returns the natural width of the image, in CSS pixels.

***imageBitmap*.height[p1279]**

Returns the natural height of the image, in CSS pixels.

An ImageBitmap[p1275] object whose [[Detached]][p124] 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[p1275] object's media data can be decoded without errors, it is said to be **fully decodable**.

An ImageBitmap[p1275] object's bitmap has an origin-clean[p715] flag, which indicates whether the bitmap is tainted by content from a different origin[p939]. The flag is initially set to true and may be changed to false by the steps of createImageBitmap()[p1276].

ImageBitmap[p1275] objects are serializable objects[p122] and transferable objects[p123].

Their serialization steps[p122], given *value* and *serialized*, are:

1. If *value*'s origin-clean[p715] flag is not set, then throw a "DataCloneError" DOMException.
2. Set *serialized*.[[BitmapData]] to a copy of *value*'s bitmap data[p1276].

Their deserialization steps[p122], given *serialized*, *value*, and *targetRealm*, are:

1. Set *value*'s bitmap data[p1276] to *serialized*.[[BitmapData]].

Their transfer steps[p123], given *value* and *dataHolder*, are:

1. If *value*'s origin-clean[p715] flag is not set, then throw a "DataCloneError" DOMException.
2. Set *dataHolder*.[[BitmapData]] to *value*'s bitmap data[p1276].
3. Unset *value*'s bitmap data[p1276].

Their transfer-receiving steps[p123], given *dataHolder* and *value*, are:

1. Set *value*'s bitmap data[p1276] to *dataHolder*.[[BitmapData]].

The createImageBitmap[p1276] method uses the **bitmap task source** to settle its returned Promise.

The **createImageBitmap(*image*, *options*)** and createImageBitmap(*image*, *sx*, *sy*, *sw*, *sh*, *options*) methods, when invoked, must run these steps:

1. If either *sw* or *sh* is given and is 0, then return a promise rejected with a RangeError.
2. If either *options*'s **resizeWidth** or *options*'s **resizeHeight** is present and is 0, then return a promise rejected with an "InvalidStateError" DOMException.
3. Check the usability of the *image* argument[p748]. If this throws an exception or returns *bad*, then return a promise rejected with an "InvalidStateError" DOMException.
4. Let *promise* be a new promise.
5. Let *imageBitmap* be a new ImageBitmap[p1275] object.
6. Switch on *image*:

↪ **img**[p359]

↪ **SVG image**

1. If *image*'s media data has no natural dimensions (e.g., it's a vector graphic with no specified content size) and either *options*'s `resizeWidth`[p1276] or *options*'s `resizeHeight`[p1276] is not present, then return a promise rejected with an `"InvalidStateError"` `DOMException`.

2. If *image*'s media data has no natural dimensions (e.g., it's a vector graphic with no specified content size), it should be rendered to a bitmap of the size specified by the `resizeWidth`[p1276] and the `resizeHeight`[p1276] options.

3. Set *imageBitmap*'s bitmap data[p1276] to a copy of *image*'s media data, cropped to the source rectangle with formatting[p1278]. If this is an animated image, *imageBitmap*'s bitmap data[p1276] 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 *image* is not origin-clean[p749], then set the origin-clean[p715] flag of *imageBitmap*'s bitmap to false.

5. Queue a global task[p1195], using the bitmap task source[p1276], to resolve *promise* with *imageBitmap*.

↪ **video**[p420]

1. If *image*'s `networkState`[p435] attribute is `NETWORK_EMPTY`[p435], then return a promise rejected with an `"InvalidStateError"` `DOMException`.

2. Set *imageBitmap*'s bitmap data[p1276] to a copy of the frame at the current playback position[p448], at the media resource[p432]'s natural width[p424] and natural height[p424] (i.e., after any aspect-ratio correction has been applied), cropped to the source rectangle with formatting[p1278].

3. If *image* is not origin-clean[p749], then set the origin-clean[p715] flag of *imageBitmap*'s bitmap to false.

4. Queue a global task[p1195], using the bitmap task source[p1276], to resolve *promise* with *imageBitmap*.

↪ **canvas**[p713]

1. Set *imageBitmap*'s bitmap data[p1276] to a copy of *image*'s bitmap data[p1276], cropped to the source rectangle with formatting[p1278].

2. Set the origin-clean[p715] flag of the *imageBitmap*'s bitmap to the same value as the origin-clean[p715] flag of *image*'s bitmap.

3. Queue a global task[p1195], using the bitmap task source[p1276], to resolve *promise* with *imageBitmap*.

↪ **Blob**

Run these steps in parallel[p44]:

1. Let *imageData* be the result of reading *image*'s data. If an error occurs during reading of the object[p63], then queue a global task[p1195], using the bitmap task source[p1276], to reject *promise* with an `"InvalidStateError"` `DOMException` and abort these steps.

2. Apply the image sniffing rules to determine the file format of *imageData*, with MIME type of *image* (as given by *image*'s `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 natural size), then queue a global task[p1195], using the bitmap task source[p1276], to reject *promise* with an `"InvalidStateError"` `DOMException` and abort these steps.

4. Set *imageBitmap*'s bitmap data[p1276] to *imageData*, cropped to the source rectangle with formatting[p1278]. If this is an animated image, *imageBitmap*'s bitmap data[p1276] 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. Queue a global task[p1195], using the bitmap task source[p1276], to resolve *promise* with *imageBitmap*.

↪ **ImageData**[p1272]

1. Let *buffer* be *image*'s `data`[p1274] attribute value's [[ViewedArrayBuffer]] internal slot.

2. If IsDetachedBuffer(*buffer*) is true, then return a promise rejected with an "InvalidStateError" DOMException.

3. Set *imageBitmap*'s bitmap data[p1276] to *image*'s image data, cropped to the source rectangle with formatting[p1278].

4. Queue a global task[p1195], using the bitmap task source[p1276], to resolve *promise* with *imageBitmap*.

↪ **ImageBitmap**[p1275]

1. Set *imageBitmap*'s bitmap data[p1276] to a copy of *image*'s bitmap data[p1276], cropped to the source rectangle with formatting[p1278].

2. Set the origin-clean[p715] flag of *imageBitmap*'s bitmap to the same value as the origin-clean[p715] flag of *image*'s bitmap.

3. Queue a global task[p1195], using the bitmap task source[p1276], to resolve *promise* with *imageBitmap*.

↪ **VideoFrame**

1. Set *imageBitmap*'s bitmap data[p1276] to a copy of *image*'s visible pixel data, cropped to the source rectangle with formatting[p1278].

2. Queue a global task[p1195], using the bitmap task source[p1276], to resolve *promise* with *imageBitmap*.

7. Return *promise*.

When the steps above require that the user agent **crop bitmap data to the source rectangle with formatting**, the user agent must run the following steps:

1. Let *input* be the bitmap data[p1276] being transformed.

2. If *sx*, *sy*, *sw* and *sh* are specified, let *sourceRectangle* be a rectangle whose corners are the four points (*sx*, *sy*), (*sx*+*sw*, *sy*), (*sx*+*sw*, *sy*+*sh*), (*sx*, *sy*+*sh*). Otherwise, let *sourceRectangle* be a rectangle whose corners are the four points (0, 0), (width of *input*, 0), (width of *input*, height of *input*), (0, height of *input*).

Note

If either sw *or* sh *are negative, then the top-left corner of this rectangle will be to the left or above the* (sx, sy) *point.*

3. Let *outputWidth* be determined as follows:

↪ **If the resizeWidth[p1276] member of *options* is specified**
the value of the resizeWidth[p1276] member of *options*

↪ **If the resizeWidth[p1276] member of *options* is not specified, but the resizeHeight[p1276] member is specified**
the width of *sourceRectangle*, times the value of the resizeHeight[p1276] member of *options*, divided by the height of *sourceRectangle*, rounded up to the nearest integer

↪ **If neither resizeWidth[p1276] nor resizeHeight[p1276] are specified**
the width of *sourceRectangle*

4. Let *outputHeight* be determined as follows:

↪ **If the resizeHeight[p1276] member of *options* is specified**
the value of the resizeHeight[p1276] member of *options*

↪ **If the resizeHeight[p1276] member of *options* is not specified, but the resizeWidth[p1276] member is specified**
the height of *sourceRectangle*, times the value of the resizeWidth[p1276] member of *options*, divided by the width of *sourceRectangle*, rounded up to the nearest integer

↪ **If neither resizeWidth[p1276] nor resizeHeight[p1276] 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][p1582]

   Note

   *If the value is "**from-image**", no extra step is needed.*

   Note

   *There used to be a "**none**" enum value. It was renamed to "from-image[p1279]". In the future, "none[p1279]" will be added back with a different meaning.*

9. If *image* is an img[p359] 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 default 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.

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[p785] 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[p785].

11. Return *output*.

The **close()** method steps are:

1. Set this's [[Detached]][p124] internal slot value to true.

2. Unset this's bitmap data[p1276].

The **width** getter steps are:

1. If this's [[Detached]][p124] internal slot's value is true, then return 0.

2. Return this's width, in CSS pixels.

The **height** getter steps are:

1. If this's [[Detached]][p124] internal slot's value is true, then return 0.

2. Return this's height, in CSS pixels.

The ResizeQuality[p1275] enumeration is used to express a preference for the interpolation quality to use when scaling images.

The "**pixelated**" value indicates a preference for scaling the image to preserve the pixelation of the original as much as possible, with minor smoothing as necessary to avoid distorting the image when the target size is not a clean multiple of the original.

To implement "pixelated[p1279]", for each axis independently, first determine the integer multiple of its natural size that puts it closest to the target size and is greater than zero. Scale it to this integer-multiple-size using nearest neighbor, then scale it the rest of the way to the target size using 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, except for "pixelated[p1279]" as described above.

Example

Using this API, a sprite sheet can be precut and prepared:

```
var sprites = {};
function loadMySprites() {
  var image = new Image();
  image.src = 'mysprites.png';
  var resolver;
  var promise = new Promise(function (arg) { resolver = arg });
  image.onload = function () {
    resolver(Promise.all([
      createImageBitmap(image,  0,  0, 40, 40).then(function (image) { sprites.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);
```

8.12 Animation frames §[p1280] 80

Some objects include the AnimationFrameProvider[p1280] interface mixin.

IDL

```
callback FrameRequestCallback = undefined (DOMHighResTimeStamp time);

interface mixin AnimationFrameProvider {
  unsigned long requestAnimationFrame(FrameRequestCallback callback);
  undefined cancelAnimationFrame(unsigned long handle);
};
Window includes AnimationFrameProvider;
DedicatedWorkerGlobalScope includes AnimationFrameProvider;
```

Each AnimationFrameProvider[p1280] object also has a **target object** that stores the provider's internal state. It is defined as follows:

If the AnimationFrameProvider[p1280] is a Window[p965]

The Window[p965]'s associated Document[p966]

If the AnimationFrameProvider[p1280] is a DedicatedWorkerGlobalScope[p1324]

The DedicatedWorkerGlobalScope[p1324]

Each target object[p1280] 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[p1280] *provider* is considered **supported** if any of the following are true:

- *provider* is a Window[p965].
- *provider*'s owner set[p1322] contains a Document[p135] object.
- Any of the DedicatedWorkerGlobalScope[p1324] objects in *provider*'s owner set[p1322] are supported[p1281].

The **requestAnimationFrame(*callback*)** method steps are:

1. If this is not supported[p1281], then throw a "NotSupportedError" DOMException.
2. Let *target* be this's target object[p1280].
3. Increment *target*'s animation frame callback identifier[p1281] by one, and let *handle* be the result.
4. Let *callbacks* be *target*'s map of animation frame callbacks[p1281].
5. Set *callbacks*[*handle*] to *callback*.
6. Return *handle*.

The **cancelAnimationFrame(*handle*)** method steps are:

1. If this is not supported[p1281], then throw a "NotSupportedError" DOMException.
2. Let *callbacks* be this's target object[p1280]'s map of animation frame callbacks[p1281].
3. Remove *callbacks*[*handle*].

To **run the animation frame callbacks** for a target object[p1280] *target* with a timestamp *now*:

1. Let *callbacks* be *target*'s map of animation frame callbacks[p1281].
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* with « *now* » and "report".

Example

Inside workers, requestAnimationFrame()[p1281] can be used together with an OffscreenCanvas[p777] transferred from a canvas[p713] element. First, in the document, transfer control to the worker:

```
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:

```
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 §p12 83

¶p12 83 **Note**

The WebSocket *interface used to be defined here. It is now defined in* WebSockets. *[WEBSOCKETS]p1588*

9.1 The MessageEventp1283 interface §p12 83

Messages in server-sent eventsp1284, cross-document messagingp1293, channel messagingp1296, broadcast channelsp1303, and *WebSockets* use the MessageEventp1283 interface for their messagep1575 events: [WEBSOCKETS]p1588

IDL

```
[Exposed=(Window,Worker,AudioWorklet)]
interface MessageEvent : Event {
  constructor(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;
```

Each MessageEventp1283 has an **origin** (an originp939, a string, or null), initially null.

For web developers (non-normative)

***event*.data**p1284

Returns the data of the message.

***event*.origin**p1284

Returns the origin of the message, for server-sent eventsp1284 and cross-document messagingp1293.

***event*.lastEventId**p1284

Returns the last event ID stringp1286, for server-sent eventsp1284.

***event*.source**p1284

Returns the WindowProxyp977 of the source window, for cross-document messagingp1293, and the MessagePortp1299 being attached, in the connectp1575 event fired at SharedWorkerGlobalScopep1324 objects.

***event*.ports**p1284

Returns the MessagePortp1299 array sent with the message, for cross-document messagingp1293 and channel messagingp1296.

The **data** attribute must return the value it was initialized to. It represents the message being sent.

The **origin** attribute represents, in server-sent events[p1284] and cross-document messaging[p1293], 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 origin[p1284] getter steps are:

1. If this's origin[p1283] is an origin[p939], then return the serialization[p939] of this's origin[p1283].
2. If this's origin[p1283] is null, then return the empty string.
3. Return this's origin[p1283].

When the origin[p1284] attribute is "initialized" (during MessageEvent[p1283]'s constructor, for example), the initialization value is placed into the object's origin[p1283].

Objects implementing the MessageEvent[p1283] interface's extract an origin[p943] steps are to return this's origin[p1283] if it is an origin[p939]; otherwise null.

The **lastEventId** attribute must return the value it was initialized to. It represents, in server-sent events[p1284], the last event ID string[p1286] of the event source.

The **source** attribute must return the value it was initialized to. It represents, in cross-document messaging[p1293], the WindowProxy[p977] of the browsing context[p1046] of the Window[p965] object from which the message came; and in the connect[p1575] events used by shared workers[p1324], the newly connecting MessagePort[p1299].

The **ports** attribute must return the value it was initialized to. It represents, in cross-document messaging[p1293] and channel messaging[p1296], the MessagePort[p1299] array being sent.

The **initMessageEvent(*type*, *bubbles*, *cancelable*, *data*, *origin*, *lastEventId*, *source*, *ports*)** method must initialize the event in a manner analogous to the similarly-named initEvent() method. [DOM][p1582]

Note

Various APIs (e.g., WebSocket, EventSource[p1285]) use the MessageEvent[p1283] interface for their message[p1575] event without using the MessagePort[p1299] API.

9.2 Server-sent events §[p12 84]

9.2.1 Introduction §[p12 84]

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[p1285] interface.

Using this API consists of creating an EventSource[p1285] object and registering an event listener.

```
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[p1552] MIME type:

```
data: This is the first message.

data: This is the second message, it
```

```
data: 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):

```
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[p404] 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[p1292].

9.2.2 The EventSource[p1285] interface §[p12 85]

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
  attribute EventHandler onopen;
  attribute EventHandler onmessage;
  attribute EventHandler onerror;
  undefined close();
};

dictionary EventSourceInit {
  boolean withCredentials = false;
```

```
};
```

Each EventSource[p1285] 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[p1286] these are not currently exposed on the EventSource[p1285] object.

For web developers (non-normative)

***source* = new EventSource[p1286](*url* [, { withCredentials[p1285]: true }])**

Creates a new EventSource[p1285] object.

url is a string giving the URL that will provide the event stream.

Setting withCredentials[p1285] to true will set the credentials mode for connection requests to *url* to "include".

***source*.close[p1287]()**

Aborts any instances of the fetch algorithm started for this EventSource[p1285] object, and sets the readyState[p1287] attribute to CLOSED[p1287].

***source*.url[p1287]**

Returns the URL providing the event stream[p1286].

***source*.withCredentials[p1287]**

Returns true if the credentials mode for connection requests to the URL providing the event stream[p1286] is set to "include", and false otherwise.

***source*.readyState[p1287]**

Returns the state of this EventSource[p1285] 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[p1285] object.
2. Let *settings* be *ev*'s relevant settings object[p1151].
3. Let *urlRecord* be the result of encoding-parsing a URL[p101] given *url*, relative to *settings*.
4. If *urlRecord* is failure, then throw a "SyntaxError" DOMException.
5. Set *ev*'s url[p1286] to *urlRecord*.
6. Let *corsAttributeState* be Anonymous[p103].
7. If the value of *eventSourceInitDict*'s withCredentials[p1285] member is true, then set *corsAttributeState* to Use Credentials[p103] and set *ev*'s withCredentials[p1287] attribute to true.
8. Let *request* be the result of creating a potential-CORS request[p102] given *urlRecord*, the empty string, and *corsAttributeState*.
9. Set *request*'s client to *settings*.
10. User agents may set (`Accept`, `text/event-stream`[p1552]) in *request*'s header list.
11. Set *request*'s cache mode to "no-store".
12. Set *request*'s initiator type to "other".
13. Set *ev*'s request[p1286] to *request*.
14. Let *processEventSourceEndOfBody* given response *res* be the following step: if *res* is not a network error, then reestablish the connection[p1287].

15. Fetch *request*, with *processResponseEndOfBody* set to *processEventSourceEndOfBody* and *processResponse* set to the following steps given response *res*:

    1. If *res* is an aborted network error, then fail the connection[p1288].
    2. Otherwise, if *res* is a network error, then reestablish the connection[p1287], unless the user agent knows that to be futile, in which case the user agent may fail the connection[p1288].
    3. Otherwise, if *res*'s status is not 200, or if *res*'s `Content-Type[p102]` is not `text/event-stream[p1552]`, then fail the connection[p1288].
    4. Otherwise, announce the connection[p1287] and interpret[p1289] *res*'s body line by line.

16. Return *ev*.

The **url** attribute's getter must return the serialization of this EventSource[p1285] object's url[p1286].

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()[p1287] method was invoked.

When the object is created, its readyState[p1287] must be set to CONNECTING[p1287] (0). The rules given below for handling the connection define when the value changes.

The **close()** method must abort any instances of the fetch algorithm started for this EventSource[p1285] object, and must set the readyState[p1287] attribute to CLOSED[p1287].

The following are the event handlers[p1206] (and their corresponding event handler event types[p1209]) that must be supported, as event handler IDL attributes[p1207], by all objects implementing the EventSource[p1285] interface:

Event handler[p1206]	Event handler event type[p1209]
onopen	open[p1576]
onmessage	message[p1575]
onerror	error[p1575]

9.2.3 Processing model §[p1287]

When a user agent is to **announce the connection**, the user agent must queue a task[p1194] which, if the readyState[p1287] attribute is set to a value other than CLOSED[p1287], sets the readyState[p1287] attribute to OPEN[p1287] and fires an event named open[p1576] at the EventSource[p1285] object.

When a user agent is to **reestablish the connection**, the user agent must run the following steps. These steps are run in parallel[p44], not as part of a task[p1194]. (The tasks that it queues, of course, are run like normal tasks and not themselves in parallel[p44].)

1. Queue a task[p1194] to run the following steps:
    1. If the readyState[p1287] attribute is set to CLOSED[p1287], abort the task.
    2. Set the readyState[p1287] attribute to CONNECTING[p1287].
    3. Fire an event named error[p1575] at the EventSource[p1285] 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[p1194] to run the following steps:

    1. If the `EventSource`[p1285] object's `readyState`[p1287] attribute is not set to `CONNECTING`[p1287], then return.

    2. Let *request* be the `EventSource`[p1285] object's request[p1286].

    3. If the `EventSource`[p1285] object's last event ID string[p1286] is not the empty string:

        1. Let *lastEventIDValue* be the `EventSource`[p1285] object's last event ID string[p1286], encoded as UTF-8.

        2. Set (`` `Last-Event-ID`[p1288] ``, *lastEventIDValue*) 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[p1194] which, if the `readyState`[p1287] attribute is set to a value other than `CLOSED`[p1287], sets the `readyState`[p1287] attribute to `CLOSED`[p1287] and fires an event named `error`[p1575] at the `EventSource`[p1285] object. **Once the user agent has failed the connection[p1288], it does *not* attempt to reconnect.**

The task source[p1194] for any tasks[p1194] that are queued[p1194] by `EventSource`[p1285] objects is the **remote event task source**.

9.2.4 The `` `Last-Event-ID`[p1288] `` header §[p12] 88

The **`Last-Event-ID`** HTTP request header reports an `EventSource`[p1285] object's last event ID string[p1286] to the server when the user agent is to reestablish the connection[p1287].

See whatwg/html issue #7363 to define the value space better. It is essentially any UTF-8 encoded string, that does not contain U+0000 NULL, U+000A LF, or U+000D CR.

9.2.5 Parsing an event stream §[p12] 88

This event stream format's MIME type is `text/event-stream`[p1552].

The event stream format is as described by the `stream` production of the following ABNF, the character set for which is Unicode. [ABNF][p1579]

```
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+000A LINE FEED (LF)
cr            = %x000D ; U+000D CARRIAGE RETURN (CR)
space         = %x0020 ; U+0020 SPACE
colon         = %x003A ; U+003A COLON (:)
bom           = %xFEFF ; U+FEFF BYTE ORDER MARK
name-char     = %x0000-0009 / %x000B-000C / %x000E-0039 / %x003B-10FFFF
                ; a scalar value other than U+000A LINE FEED (LF), U+000D CARRIAGE RETURN (CR), or
U+003A COLON (:)
```

```
any-char      = %x0000-0009 / %x000B-000C / %x000E-10FFFF
                ; a scalar value other than U+000A LINE FEED (LF) or U+000D CARRIAGE RETURN (CR)
```

Event streams in this format must always be encoded as UTF-8. [ENCODING]p1582

Lines must be separated by either a U+000D CARRIAGE RETURN U+000A LINE FEED (CRLF) character pair, a single U+000A LINE FEED (LF) character, or a single U+000D 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+000A LINE FEED (LF) character is safe, block buffering or line buffering with different expected line endings can cause delays in event dispatch.

9.2.6 Interpreting an event stream §p12 89

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 eventp1290, 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 fieldp1289 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 fieldp1289 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 the 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[p1286] 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[p1290] are as follows:

1. Set the last event ID string[p1286] 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[p1286] 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`[p1283], in the relevant realm[p1151] of the `EventSource`[p1285] object.
5. Initialize *event*'s `type` attribute to "`message`[p1575]", its `data`[p1284] attribute to *data*, its origin[p1283] to the origin of the event stream's final URL (i.e., the URL after redirects), and its `lastEventId`[p1284] attribute to the last event ID string[p1286] 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[p1194] which, if the `readyState`[p1287] attribute is set to a value other than `CLOSED`[p1287], dispatches the newly created event at the `EventSource`[p1285] 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[p1286], then the event's `lastEventId`[p1284] 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[p1290] 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`[p1575] with the interface `MessageEvent`[p1283] to be dispatched on the `EventSource`[p1285] object. The event's `data`[p1284] attribute would contain the string "`YHOO\n+2\n10`" (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[p1288]` 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[p1288]` 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
```

Example

The following stream fires two events:

```
data

data
data

data:
```

The first block fires events with the data set to the empty string, as would the last block if it was followed by a blank line. The middle block fires an event with the data set to a single newline character. The last block is discarded because it is not followed by a blank line.

Example

The following stream fires two identical events:

```
data:test

data: test
```

This is because the space after the colon is ignored if present.

9.2.7 Authoring notes [§p12 91]

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.

9.2.8 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.

Example

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.

9.2.9 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 the EventSource object itself.

While there is a task queued by an EventSource[p1285] object on the remote event task source[p1288], there must be a strong reference from the Window[p965] or WorkerGlobalScope[p1322] object that the EventSource[p1285] object's constructor was invoked from to that EventSource[p1285] object.

If a user agent is to **forcibly close** an EventSource[p1285] object (this happens when a Document[p135] object goes away permanently), the user agent must abort any instances of the fetch algorithm started for this EventSource[p1285] object, and must set the readyState[p1287] attribute to CLOSED[p1287].

If an EventSource[p1285] 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[p1285].

9.2.10 Implementation advice §[p12 93]

This section is non-normative.

User agents are strongly urged to provide detailed diagnostic information about EventSource[p1285] 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[p1285] 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[p1575] event is fired, since little to no information can be made available in the events themselves.

9.3 Cross-document messaging §[p12 93]

MDN

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.

¶[p12 93] Note

The postMessage()[p1296] API can be used as a tracking vector.

9.3.1 Introduction §[p12 93]

This section is non-normative.

Example

For example, if document A contains an iframe[p404] element that contains document B, and script in document A calls postMessage()[p1296] on the Window[p965] object of document B, then a message event will be fired on that object, marked as originating from the Window[p965] of document A. The script in document A might look like:

```
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:

```
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.

9.3.2 Security §p12 94

9.3.2.1 Authors §p12 94

⚠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`p1284 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`p1284 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()`p1296 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.

9.3.2.2 User agents §p12 94

The integrity of this API is based on the inability for scripts of one origin[p939] to post arbitrary events (using `dispatchEvent()` or otherwise) to objects in other origins (those that are not the same[p940]).

Note

Implementers 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[p939], to protect naïve sites from denial-of-service attacks.

9.3.3 Posting messages §[p1295]

For web developers (non-normative)

***window*.postMessage[p1296](*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`[p1299] 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`[p966] 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[p1296](*message*, *targetOrigin* [, *transfer*])**

This is an alternate version of `postMessage()`[p1296] 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`[p965] of a browsing context[p1046] that has just been navigated to a new `Document`[p135] is likely to result in the message not receiving its intended recipient: the scripts in the target browsing context[p1046] 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`[p965] of newly created child `iframe`[p404], authors are advised to have the child `Document`[p135] 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[p1145].
2. Let *incumbentSettings* be the incumbent settings object[p1149].
3. Let *targetOrigin* be *options*["`targetOrigin`[p966]"].
4. If *targetOrigin* is a single U+002F SOLIDUS character (/), then set *targetOrigin* to *incumbentSettings*'s origin[p1144].
5. Otherwise, if *targetOrigin* is not a single U+002A ASTERISK character (*):
    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`[p1299]"].
7. Let *serializeWithTransferResult* be StructuredSerializeWithTransfer[p131](*message*, *transfer*). Rethrow any exceptions.
8. Queue a global task[p1195] 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`[p966]'s origin is not same origin[p940] with *targetOrigin*, then return.
    2. Let *origin* be *incumbentSettings*'s origin[p1144].
    3. Let *source* be the `WindowProxy`[p977] object corresponding to *incumbentSettings*'s global object[p1145] (a `Window`[p965] object).
    4. Let *deserializeRecord* be StructuredDeserializeWithTransfer[p132](*serializeWithTransferResult*, *targetRealm*).

       If this throws an exception, catch it, fire an event named `messageerror`[p1575] at *targetWindow*, using `MessageEvent`[p1283], with its origin[p1283] initialized to *origin* and the `source`[p1284] 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[p1299] objects in *deserializeRecord*.[[TransferredValues]], if any, maintaining their relative order.

7. Fire an event named message[p1575] at *targetWindow*, using MessageEvent[p1283], with its origin[p1283] initialized to *origin*, the source[p1284] attribute initialized to *source*, the data[p1284] attribute initialized to *messageClone*, and the ports[p1284] attribute initialized to *newPorts*.

The Window[p965] interface's **postMessage(*message, options*)** method steps are to run the window post message steps[p1295] given this, *message*, and *options*.

The Window[p965] interface's **postMessage(*message, targetOrigin, transfer*)** method steps are to run the window post message steps[p1295] given this, *message*, and «["targetOrigin[p966]" → *targetOrigin*, "transfer[p1299]" → *transfer*]».

9.4 Channel messaging §[p1296]

9.4.1 Introduction §[p1296]

This section is non-normative.

To enable independent pieces of code (e.g. running in different browsing contexts[p1046]) to communicate directly, authors can use channel messaging[p1296].

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[p1194].

To create a connection (two "entangled" ports), the `MessageChannel()` constructor is called:

```
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()[p1296]:

```
otherWindow.postMessage('hello', 'https://example.com', [channel.port2]);
```

To send messages, the postMessage()[p1302] method on the port is used:

```
channel.port1.postMessage('hello');
```

To receive messages, one listens to message[p1575] events:

```
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[p1299]:

```
port1.postMessage(['hello', 'world']);
```

9.4.1.1 Examples §[p1296]

This section is non-normative.

Example

In this example, two JavaScript libraries are connected to each other using MessagePort[p1299]s. This allows the libraries to later be hosted in different frames, or in Worker[p1332] objects, without any change to the APIs.

```
<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:

```
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;
      // ...
    }
  };
};
```

Alternatively, it could be implemented as follows:

```
function addContactsProvider(port) {
  port.addEventListener('message', function (event) {
    if (event.data.messageType == 'search-result')
      handleSearchResult(event.data.results);
  });
  port.addEventListener('message', function (event) {
    if (event.data.messageType == 'search-done')
      handleSearchDone();
  });
  port.addEventListener('message', function (event) {
    if (event.data.messageType == 'search-error')
      handleSearchError(event.data.message);
  });
  // ...
  port.start();
};
```

The key difference is that when using addEventListener(), the start()[p1302] method must also be invoked. When using onmessage[p1299], the call to start()[p1302] is implied.

The start()[p1302] method, whether called explicitly or implicitly (by setting onmessage[p1299]), starts the flow of messages: messages posted on message ports are initially paused, so that they don't get dropped on the floor before the script has had a chance to set up its handlers.

9.4.1.2 Ports as the basis of an object-capability model on the web §[p12 97]

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 *provider* as the only mechanism by which another origin *consumer* can effect change in or obtain information from *provider*.

For example, consider a situation in which a social web site embeds in one iframe[p404] the user's email contacts provider (an address book site, from a second origin), and in a second iframe[p404] a game (from a third origin). The outer social site and the game in the second iframe[p404] cannot access anything inside the first iframe[p404]; together they can only:

- Navigate[p1063] the iframe[p404] to a new URL, such as the same URL but with a different fragment, causing the Window[p965] in the iframe[p404] to receive a hashchange[p1575] event.
- Resize the iframe[p404], causing the Window[p965] in the iframe[p404] to receive a resize event.
- Send a message[p1575] event to the Window[p965] in the iframe[p404] using the window.postMessage()[p1296] 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[p1299] 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[p1299] 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.

9.4.1.3 Ports as the basis of abstracting out service implementations §[p12 98]

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[p404], an evolution of the service might instead want to use a shared worker[p1333] with a single WebSocket connection.

If the initial design used MessagePort[p1299] objects to grant capabilities, or even just to allow multiple simultaneous independent sessions, the service implementation can switch from the XMLHttpRequests-in-each-iframe[p404] 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.

9.4.2 Message channels §[p12 98]

IDL

```
[Exposed=(Window,Worker)]
interface MessageChannel {
  constructor();

  readonly attribute MessagePort port1;
  readonly attribute MessagePort port2;
};
```

For web developers (non-normative)

***channel* = new MessageChannel[p1299]()**

Returns a new MessageChannel[p1298] object with two new MessagePort[p1299] objects.

***channel*.port1** p1299

Returns the first MessagePort p1299 object.

***channel*.port2** p1299

Returns the second MessagePort p1299 object.

A MessageChannel p1298 object has an associated **port 1** and an associated **port 2**, both MessagePort p1299 objects.

The **new MessageChannel()** constructor steps are:

1. Set this's port 1 p1299 to a new MessagePort p1299 in this's relevant realm p1151.
2. Set this's port 2 p1299 to a new MessagePort p1299 in this's relevant realm p1151.
3. Entangle p1300 this's port 1 p1299 and this's port 2 p1299.

The **port1** getter steps are to return this's port 1 p1299.

The **port2** getter steps are to return this's port 2 p1299.

9.4.3 The MessageEventTarget p1299 mixin §p12 99

IDL

```
interface mixin MessageEventTarget {
  attribute EventHandler onmessage;
  attribute EventHandler onmessageerror;
};
```

The following are the event handlers p1206 (and their corresponding event handler event types p1209) that must be supported, as event handler IDL attributes p1207, by objects implementing the MessageEventTarget p1299 interface:

Event handler p1206	Event handler event type p1209
onmessage	message p1575
onmessageerror	messageerror p1575

9.4.4 Message ports §p12 99

Each channel has two message ports. Data sent through one port is received by the other port, and vice versa.

IDL

```
[Exposed=(Window,Worker,AudioWorklet), Transferable]
interface MessagePort : EventTarget {
  undefined postMessage(any message, sequence<object> transfer);
  undefined postMessage(any message, optional StructuredSerializeOptions options = {});
  undefined start();
  undefined close();
};

MessagePort includes MessageEventTarget;

dictionary StructuredSerializeOptions {
  sequence<object> transfer = [];
};
```

For web developers (non-normative)

***port*.postMessage[p1302](*message* [, *transfer*])**

***port*.postMessage[p1302](*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.

***port*.start[p1302]()**

Begins dispatching messages received on the port.

***port*.close[p1302]()**

Disconnects the port, so that it is no longer active.

Each MessagePort[p1299] object has a **message event target** (a MessageEventTarget[p1299]), to which the message[p1575] and messageerror[p1575] events are dispatched. Unless otherwise specified, it defaults to the MessagePort[p1299] object itself.

Each MessagePort[p1299] object can be entangled with another (a symmetric relationship). Each MessagePort[p1299] object also has a task source[p1194] called the **port message queue**, initially empty. A port message queue[p1300] 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[p1299] also has a **has been shipped** flag, which must initially be false.

When a port's port message queue[p1300] is enabled, the event loop[p1193] must use it as one of its task sources[p1194]. When a port's relevant global object[p1151] is a Window[p965], all tasks[p1194] queued[p1194] on its port message queue[p1300] must be associated with the port's relevant global object[p1151]'s associated Document[p966].

Note

If the document is fully active[p1051], but the event listeners were all created in the context of documents that are not *fully active[p1051], then the messages will not be received unless and until the documents become fully active[p1051] again.*

Each event loop[p1193] has a task source[p1194] called the **unshipped port message queue**. This is a virtual task source[p1194]: it must act as if it contained the tasks[p1194] of each port message queue[p1300] of each MessagePort[p1299] whose has been shipped[p1300] flag is false, whose port message queue[p1300] is enabled, and whose relevant agent[p1141]'s event loop[p1193] is that event loop[p1193], in the order in which they were added to their respective task source[p1194]. When a task[p1194] would be removed from the unshipped port message queue[p1300], it must instead be removed from its port message queue[p1300].

When a MessagePort[p1299]'s has been shipped[p1300] flag is false, its port message queue[p1300] must be ignored for the purposes of the event loop[p1193]. (The unshipped port message queue[p1300] is used instead.)

Note

The has been shipped[p1300] 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[p1299]'s has been shipped[p1300] flag is true, its port message queue[p1300] acts as a first-class task source[p1194], unaffected to any unshipped port message queue[p1300].

When the user agent is to **entangle** two MessagePort[p1299] 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[p1298] object, then that MessageChannel[p1298] 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[p1298] 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.

The **disentangle** steps, given a MessagePort[p1299] *initiatorPort* which initiates disentangling, are as follows:

1. Let *otherPort* be the MessagePort[p1299] which *initiatorPort* was entangled with.
2. Assert: *otherPort* exists.
3. Disentangle *initiatorPort* and *otherPort*, so that they are no longer entangled or associated with each other.

MessagePort[p1299] objects are transferable objects[p123]. Their transfer steps[p123], given *value* and *dataHolder*, are:

1. Set *value*'s has been shipped[p1300] flag to true.
2. Set *dataHolder*.[[PortMessageQueue]] to *value*'s port message queue[p1300].
3. If *value* is entangled with another port *remotePort*:
    1. Set *remotePort*'s has been shipped[p1300] flag to true.
    2. Set *dataHolder*.[[RemotePort]] to *remotePort*.
4. Otherwise, set *dataHolder*.[[RemotePort]] to null.

Their transfer-receiving steps[p123], given *dataHolder* and *value*, are:

1. Set *value*'s has been shipped[p1300] flag to true.
2. Move all the tasks[p1194] that are to fire message[p1575] events in *dataHolder*.[[PortMessageQueue]] to the port message queue[p1300] of *value*, if any, leaving *value*'s port message queue[p1300] in its initial disabled state, and, if *value*'s relevant global object[p1151] is a Window[p965], associating the moved tasks[p1194] with *value*'s relevant global object[p1151]'s associated Document[p966].
3. If *dataHolder*.[[RemotePort]] is not null, then entangle[p1300] *dataHolder*.[[RemotePort]] and *value*. (This will disentangle *dataHolder*.[[RemotePort]] from the original port that was transferred.)

The **message port post message steps**, given *sourcePort*, *targetPort*, *message*, and *options* are as follows:

1. Let *transfer* be *options*["transfer[p1299]"].
2. If *transfer* contains *sourcePort*, 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[p131](*message*, *transfer*). Rethrow any exceptions.
6. If *targetPort* is null, or if *doomed* is true, then return.
7. Add a task[p1194] that runs the following steps to the port message queue[p1300] of *targetPort*:
    1. Let *finalTargetPort* be the MessagePort[p1299] in whose port message queue[p1300] 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 *messageEventTarget* be *finalTargetPort*'s message event target[p1300].
    3. Let *targetRealm* be *finalTargetPort*'s relevant realm[p1151].
    4. Let *deserializeRecord* be StructuredDeserializeWithTransfer[p132](*serializeWithTransferResult*, *targetRealm*).

    If this throws an exception, catch it, fire an event named messageerror[p1575] at *messageEventTarget*, using

MessageEvent[p1283], and then return.

5. Let *messageClone* be *deserializeRecord*.[[Deserialized]].

6. Let *newPorts* be a new frozen array consisting of all MessagePort[p1299] objects in *deserializeRecord*.[[TransferredValues]], if any, maintaining their relative order.

7. Fire an event named message[p1575] at *messageEventTarget*, using MessageEvent[p1283], with the data[p1284] attribute initialized to *messageClone* and the ports[p1284] attribute initialized to *newPorts*.

The **postMessage(*message, options*)** method steps are:

1. Let *targetPort* be the port with which this is entangled, if any; otherwise let it be null.

2. Run the message port post message steps[p1301] providing this, *targetPort*, *message* and *options*.

The **postMessage(*message, transfer*)** method steps are:

1. Let *targetPort* be the port with which this is entangled, if any; otherwise let it be null.

2. Let *options* be «["transfer[p1299]" → *transfer*]».

3. Run the message port post message steps[p1301] providing this, *targetPort*, *message* and *options*.

The **start()** method steps are to enable this's port message queue[p1300], if it is not already enabled.

The **close()** method steps are:

1. Set this's [[Detached]][p124] internal slot value to true.

2. If this is entangled, disentangle[p1301] it.

The first time a MessagePort[p1299] object's onmessage[p1299] IDL attribute is set, the port's port message queue[p1300] must be enabled, as if the start()[p1302] method had been called.

9.4.5 Broadcasting to many ports §[p1302]

This section is non-normative.

Broadcasting to many ports is in principle relatively simple: keep an array of MessagePort[p1299] 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[p1299] object, and let it be garbage collected.

9.4.6 Ports and garbage collection §[p1302]

When a MessagePort[p1299] object *o* is entangled, user agents must either act as if *o*'s entangled MessagePort[p1299] object has a strong reference to *o*, or as if *o*'s relevant global object[p1151] has a strong reference to *o*.

Note

Thus, a message port can be received, given an event listener, and then forgotten, and so long as that event listener could receive a message, the channel will be maintained.

Of course, if this was to occur on both sides of the channel, then both ports could be garbage collected, since they would not be reachable from live code, despite having a strong reference to each other.

Furthermore, a MessagePort[p1299] object must not be garbage collected while there exists an event referenced by a task[p1194] in a task queue[p1193] that is to be dispatched on that MessagePort[p1299] object, or while the MessagePort[p1299] object's port message queue[p1300] is enabled and not empty.

Note

Authors are strongly encouraged to explicitly close MessagePort[p1299] objects to disentangle them, so that their resources can be recollected. Creating many MessagePort[p1299] 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[p1299]s where garbage collection can involve cross-process coordination.

9.5 Broadcasting to other browsing contexts §[p1303]

Pages on a single origin[p939] opened by the same user in the same user agent but in different unrelated browsing contexts[p1046] 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[p1333] 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

```
[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[p1303](*name*)**

Returns a new BroadcastChannel[p1303] object via which messages for the given channel name can be sent and received.

***broadcastChannel*.name[p1303]**

Returns the channel name (as passed to the constructor).

***broadcastChannel*.postMessage[p1304](*message*)**

Sends the given message to other BroadcastChannel[p1303] objects set up for this channel. Messages can be structured objects, e.g. nested objects and arrays.

***broadcastChannel*.close[p1304]()**

Closes the BroadcastChannel[p1303] object, opening it up to garbage collection.

A BroadcastChannel[p1303] object has a **channel name** and a **closed flag**.

The **new BroadcastChannel(*name*)** constructor steps are:

1. Set this's channel name[p1303] to *name*.
2. Set this's closed flag[p1303] to false.

The **name** getter steps are to return this's channel name[p1303].

A BroadcastChannel[p1303] object is said to be **eligible for messaging** when its relevant global object[p1151] is either:

- a Window[p965] object whose associated Document[p966] is fully active[p1051], or

- a `WorkerGlobalScope`[p1322] object whose closing[p1325] flag is false and is not suspendable[p1327].

The **`postMessage(message)`** method steps are:

1. If this is not eligible for messaging[p1303], then return.
2. If this's closed flag[p1303] is true, then throw an `"InvalidStateError"` `DOMException`.
3. Let *serialized* be StructuredSerialize[p128](*message*). Rethrow any exceptions.
4. Let *sourceOrigin* be this's relevant settings object[p1151]'s origin[p1144].
5. Let *sourceStorageKey* be the result of running obtain a storage key for non-storage purposes with this's relevant settings object[p1151].
6. Let *destinations* be a list of `BroadcastChannel`[p1303] objects that match the following criteria:
    - They are eligible for messaging[p1303].
    - The result of running obtain a storage key for non-storage purposes with their relevant settings object[p1151] equals *sourceStorageKey*.
    - Their channel name[p1303] is this's channel name[p1303].
7. Remove *source* from *destinations*.
8. Sort *destinations* such that all `BroadcastChannel`[p1303] objects whose relevant agents[p1141] 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.)
9. For each *destination* in *destinations*, queue a global task[p1195] on the DOM manipulation task source[p1203] given *destination*'s relevant global object[p1151] to perform the following steps:
    1. If *destination*'s closed flag[p1303] is true, then abort these steps.
    2. Let *targetRealm* be *destination*'s relevant realm[p1151].
    3. Let *data* be StructuredDeserialize[p128](*serialized*, *targetRealm*).

       If this throws an exception, catch it, fire an event named `messageerror`[p1575] at *destination*, using `MessageEvent`[p1283], with its origin[p1283] initialized to *sourceOrigin*, and then abort these steps.
    4. Fire an event named `message`[p1575] at *destination*, using `MessageEvent`[p1283], with the `data`[p1284] attribute initialized to *data* and its origin[p1283] initialized to *sourceOrigin*.

While a `BroadcastChannel`[p1303] object whose closed flag[p1303] is false has an event listener registered for `message`[p1575] or `messageerror`[p1575] events, there must be a strong reference from the `BroadcastChannel`[p1303] object's relevant global object[p1151] to the `BroadcastChannel`[p1303] object itself.

The **`close()`** method steps are to set this's closed flag[p1303] to true.

Note

Authors are strongly encouraged to explicitly close `BroadcastChannel`[p1303] objects when they are no longer needed, so that they can be garbage collected. Creating many `BroadcastChannel`[p1303] 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[p1206] (and their corresponding event handler event types[p1209]) that must be supported, as event handler IDL attributes[p1207], by all objects implementing the `BroadcastChannel`[p1303] interface:

Event handler[p1206]	Event handler event type[p1209]
`onmessage`	`message`[p1575]
`onmessageerror`	`messageerror`[p1575]

Example

Suppose a page wants to know when the user logs out, even when the user does so from another tab at the same site:

```
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 §p13 06

10.1 Introduction §p13 06

10.1.1 Scope §p13 06

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 §p13 06

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 §p13 06

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:

```
<!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() p1333 constructor call creates a worker and returns a Worker p1332 object representing that worker, which is used to

communicate with the worker. That object's onmessage[p1299] event handler allows the code to receive messages from the worker.

The worker itself is as follows:

```
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()[p1324] 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 §[p13 07]

This section is non-normative.

All of our examples so far show workers that run classic scripts[p1153]. Workers can instead be instantiated using module scripts[p1153], 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()[p1336] 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:

```
<!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>
```

```
<div id="output"></div>

<script type="module">
  const worker = new Worker("worker.js", { type: "module" });
  worker.onmessage = receiveFromWorker;

  const url = document.querySelector("#image-url");
  const filter = document.querySelector("#filter");
  const output = document.querySelector("#output");

  url.oninput = updateImage;
  filter.oninput = sendToWorker;

  let imageData, context;

  function updateImage() {
    const img = new Image();
    img.src = url.value;

    img.onload = () => {
      const canvas = document.createElement("canvas");
      canvas.width = img.width;
      canvas.height = img.height;

      context = canvas.getContext("2d");
      context.drawImage(img, 0, 0);
      imageData = context.getImageData(0, 0, canvas.width, canvas.height);

      sendToWorker();
      output.replaceChildren(canvas);
    };
  }

  function sendToWorker() {
    worker.postMessage({ imageData, filter: filter.value });
  }

  function receiveFromWorker(e) {
    context.putImageData(e.data, 0, 0);
  }
</script>
```

The worker file is then:

```
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`:

```
export function none() {}

export function grayscale({ data: d }) {
  for (let i = 0; i < d.length; i += 4) {
    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 §p13 09

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:

```
<!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) { // note: not worker.onmessage!
    log.textContent += '\n' + e.data;
  }
</script>
```

Here is the JavaScript worker:

```
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 p1207, 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:

```
<!DOCTYPE HTML>
<html lang="en">
<meta charset="utf-8">
<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[p404] on the first page, but the same principle would apply to an entirely separate page in a separate top-level traversable[p1037].

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();
  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:

```
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 §p13 11

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:

```
<!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:

```
<!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 + '> ';
      p.appendChild(n);
      var t = document.createElement('span');
      t.textContent = message;
      p.appendChild(t);
      div.appendChild(p);
```

```
    };
    port.onmessage = function (event) {
      addMessage(name, event.data);
    };
    li.appendChild(div);
    var form = document.createElement('form');
    var p = document.createElement('p');
    var input = document.createElement('input');
    input.size = 50;
    p.appendChild(input);
    p.appendChild(document.createTextNode(' '));
    var button = document.createElement('button');
    button.textContent = 'Post';
    p.appendChild(button);
    form.onsubmit = function () {
      port.postMessage(input.value);
      addMessage('me', input.value);
      input.value = '';
      return false;
    };
    form.appendChild(p);
    li.appendChild(form);
    ul.appendChild(li);
  }
  worker.port.addEventListener('message', startPrivateChat, false);

  worker.port.start();
 </script>
</head>
<body>
 <h1>Viewer</h1>
 <h2>Map</h2>
 <p><canvas id="map" height=150 width=150></canvas></p>
 <p>
  <button type=button onclick="worker.port.postMessage('mov left')">Left</button>
  <button type=button onclick="worker.port.postMessage('mov up')">Up</button>
  <button type=button onclick="worker.port.postMessage('mov down')">Down</button>
  <button type=button onclick="worker.port.postMessage('mov right')">Right</button>
  <button type=button onclick="worker.port.postMessage('set 0')">Set 0</button>
  <button type=button onclick="worker.port.postMessage('set 1')">Set 1</button>
 </p>
 <h2>Public Chat</h2>
 <div id="public"></div>
 <form onsubmit="worker.port.postMessage('txt ' + message.value); message.value = ''; return false;">
  <p>
   <input type="text" name="message" size="50">
   <button>Post</button>
  </p>
 </form>
 <h2>Private Chat</h2>
 <ul id="private"></ul>
</body>
</html>
```

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 `configure()` method in this example.

Finally, the worker:

```
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, 0, 1, 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;
  event.ports[0].postMessage('cfg ' + name);
  event.ports[0].onmessage = getMessage;
  sendMapData(event.ports[0]._data);
};

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[p1325] 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 §[p13 15]

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:

```
<!DOCTYPE HTML>
<html lang="en">
 <head>
  <meta charset="utf-8">
  <title>Worker example: Multicore computation</title>
```

```
 </head>
 <body>
  <p>Result: <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 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) {
    // 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 §[p13 17]

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:

```
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);
     port.onmessage = function (e) {
       document.getElementById('input').value = e.data;
       port.close();
     };
   }
```

```
   function dec() {
     const port = startConversation(cryptoLib, "decrypt");
     port.postMessage(document.getElementById('private').value);
     port.postMessage(document.getElementById('input').value);
     port.onmessage = function (e) {
       document.getElementById('input').value = e.data;
       port.close();
     };
   }
  </script>
  <style>
   textarea { display: block; }
  </style>
 </head>
 <body onload="getKeys()">
  <fieldset>
   <legend>Keys</legend>
   <p><label>Public Key: <textarea id="public"></textarea></label></p>
   <p><label>Private Key: <textarea id="private"></textarea></label></p>
  </fieldset>
  <p><label>Input: <textarea id="input"></textarea></label></p>
  <p><button onclick="enc()">Encrypt</button> <button onclick="dec()">Decrypt</button></p>
 </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:

```
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 encrypt(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:

```
onmessage = function (e) {
  var k = _generateKeyPair();
  e.ports[0].postMessage(k[0]);
  e.ports[0].postMessage(k[1]);
  close();
}

function _generateKeyPair() {
  return [Math.random(), Math.random()];
}
```

For encrypting:

```
onmessage = function (e) {
  var key = e.data;
  e.ports[0].onmessage = function (e) {
    var s = e.data;
    postMessage(_encrypt(key, s));
  }
}

function _encrypt(k, s) {
  return 'encrypted-' + k + ' ' + s;
}
```

For decrypting:

```
onmessage = function (e) {
  var key = e.data;
  e.ports[0].onmessage = function (e) {
    var s = e.data;
    postMessage(_decrypt(key, s));
  }
}

function _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 §p13 20

10.1.3.1 Creating a dedicated worker §p13 20

This section is non-normative.

Creating a worker requires a URL to a JavaScript file. The `Worker()`p1333 constructor is invoked with the URL to that file as its only argument; a worker is then created and returned:

```
var worker = new Worker('helper.js');
```

If you want your worker script to be interpreted as a module scriptp1153 instead of the default classic scriptp1153, you need to use a

slightly different signature:

```
var worker = new Worker('helper.mjs', { type: "module" });
```

10.1.3.2 Communicating with a dedicated worker §[p1321]

This section is non-normative.

Dedicated workers use MessagePort[p1299] 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[p1299] event handler IDL attribute[p1207] on the Worker[p1332] object:

```
worker.onmessage = function (event) { ... };
```

You can also use the addEventListener() method.

Note

The implicit MessagePort[p1299] used by dedicated workers has its port message queue[p1300] implicitly enabled when it is created, so there is no equivalent to the MessagePort[p1299] interface's start()[p1302] method on the Worker[p1332] interface.

To *send* data to a worker, use the postMessage()[p1332] 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.

```
worker.postMessage({
  operation: 'find-edges',
  input: buffer, // an ArrayBuffer object
  threshold: 0.6,
}, [buffer]);
```

To receive a message inside the worker, the onmessage[p1299] event handler IDL attribute[p1207] is used.

```
onmessage = function (event) { ... };
```

You can again also use the addEventListener() method.

In either case, the data is provided in the event object's data[p1284] attribute.

To send messages back, you again use postMessage()[p1324]. It supports the structured data in the same manner.

```
postMessage(event.data.input, [event.data.input]); // transfer the buffer back
```

✓ MDN

10.1.3.3 Shared workers §[p1321]

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[p939]. 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()[p1334] 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.

```
var worker = new SharedWorker('service.js');
```

Communicating with shared workers is done with explicit MessagePort[p1299] objects. The object returned by the SharedWorker()[p1334] constructor holds a reference to the port on its port[p1334] attribute.

```
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[p1575] event. The port for the new client is given by the event object's source[p1284] attribute.

```
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 §[p13] 22

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[p939] can obtain a reference to that worker and communicate with it. *Service Workers* defines a third kind. [SW][p1587]

10.2.1 The global scope §[p13] 22

The global scope is the "inside" of a worker.

10.2.1.1 The WorkerGlobalScope[p1322] common interface §[p13] 22

IDL

```
[Exposed=Worker]
interface WorkerGlobalScope : EventTarget {
  readonly attribute WorkerGlobalScope self;
  readonly attribute WorkerLocation location;
  readonly attribute WorkerNavigator navigator;
  undefined importScripts((TrustedScriptURL or USVString)... urls);

  attribute OnErrorEventHandler onerror;
  attribute EventHandler onlanguagechange;
  attribute EventHandler onoffline;
  attribute EventHandler ononline;
  attribute EventHandler onrejectionhandled;
  attribute EventHandler onunhandledrejection;
};
```

WorkerGlobalScope[p1322] serves as the base class for specific types of worker global scope objects, including DedicatedWorkerGlobalScope[p1324], SharedWorkerGlobalScope[p1324], and ServiceWorkerGlobalScope.

A WorkerGlobalScope[p1322] object has an associated **owner set** (a set of Document[p135] and WorkerGlobalScope[p1322] 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[p1324] objects.

A WorkerGlobalScope[p1322] object has an associated **type** ("classic" or "module"). It is set during creation.

A WorkerGlobalScope[p1322] object has an associated **url** (null or a URL). It is initially null.

A WorkerGlobalScope[p1322] object has an associated **name** (a string). It is set during creation.

Note

The name[p1323] can have different semantics for each subclass of WorkerGlobalScope[p1322]. For DedicatedWorkerGlobalScope[p1324] instances, it is simply a developer-supplied name, useful mostly for debugging purposes. For SharedWorkerGlobalScope[p1324] instances, it allows obtaining a reference to a common shared worker via the SharedWorker()[p1334] constructor. For ServiceWorkerGlobalScope objects, it doesn't make sense (and as such isn't exposed through the JavaScript API at all).

A WorkerGlobalScope[p1322] object has an associated **policy container** (a policy container[p960]). It is initially a new policy container[p960].

A WorkerGlobalScope[p1322] object has an associated **embedder policy** (an embedder policy[p955]).

A WorkerGlobalScope[p1322] object has an associated **module map**. It is a module map[p1189], initially empty.

A WorkerGlobalScope[p1322] object has an associated **cross-origin isolated capability** boolean. It is initially false.

For web developers (non-normative)

***workerGlobal*.self**[p1323]

Returns *workerGlobal*.

***workerGlobal*.location**[p1323]

Returns *workerGlobal*'s WorkerLocation[p1337] object.

***workerGlobal*.navigator**[p1337]

Returns *workerGlobal*'s WorkerNavigator[p1337] object.

***workerGlobal*.importScripts**[p1336]**(...*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[p1322] object itself.

The **location** attribute must return the WorkerLocation[p1337] object whose associated WorkerGlobalScope object[p1337] is the WorkerGlobalScope[p1322] object.

Note

While the WorkerLocation[p1337] object is created after the WorkerGlobalScope[p1322] object, this is not problematic as it cannot be observed from script.

The following are the event handlers[p1206] (and their corresponding event handler event types[p1209]) that must be supported, as event handler IDL attributes[p1207], by objects implementing the WorkerGlobalScope[p1322] interface:

Event handler[p1206]	Event handler event type[p1209]
onerror	error[p1575]
onlanguagechange	languagechange[p1575]
onoffline	offline[p1576]
ononline	online[p1576]
onrejectionhandled	rejectionhandled[p1576]
onunhandledrejection	unhandledrejection[p1576]

10.2.1.2 Dedicated workers and the DedicatedWorkerGlobalScope[p1324] interface §[p13 23]

IDL

```
[Global=(Worker,DedicatedWorker),Exposed=DedicatedWorker]
```

```
interface DedicatedWorkerGlobalScope : WorkerGlobalScope {
  [Replaceable] readonly attribute DOMString name;

  undefined postMessage(any message, sequence<object> transfer);
  undefined postMessage(any message, optional StructuredSerializeOptions options = {});

  undefined close();
};

DedicatedWorkerGlobalScope includes MessageEventTarget;
```

DedicatedWorkerGlobalScope[p1324] objects have an associated **inside port** (a MessagePort[p1299]). 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[p1324] object.

For web developers (non-normative)

***dedicatedWorkerGlobal*.name[p1324]**

Returns *dedicatedWorkerGlobal*'s name[p1323], i.e. the value given to the Worker[p1332] constructor. Primarily useful for debugging.

***dedicatedWorkerGlobal*.postMessage[p1324](*message* [, *transfer*])**

***dedicatedWorkerGlobal*.postMessage[p1324](*message* [, { transfer[p1299] }])**

Clones *message* and transmits it to the Worker[p1332] object associated with *dedicatedWorkerGlobal*. *transfer* can be passed as a list of objects that are to be transferred rather than cloned.

***dedicatedWorkerGlobal*.close[p1324]()**

Aborts *dedicatedWorkerGlobal*.

The **name** getter steps are to return this's name[p1323]. Its value represents the name given to the worker using the Worker[p1332] constructor, used primarily for debugging purposes.

The **postMessage(*message, transfer*)** and **postMessage(*message, options*)** methods on DedicatedWorkerGlobalScope[p1324] objects act as if, when invoked, it immediately invoked the respective postMessage(*message, transfer*)[p1302] and postMessage(*message, options*)[p1302] 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[p1194] that have been added to *workerGlobal*'s relevant agent[p1141]'s event loop[p1193]'s task queues[p1193].
2. Set *workerGlobal*'s closing[p1325] flag to true. (This prevents any further tasks from being queued.)

The **close()** method steps are to close a worker[p1324] given this.

✓ MDN

10.2.1.3 Shared workers and the SharedWorkerGlobalScope[p1324] interface §[p13 24]

IDL

```
[Global=(Worker,SharedWorker),Exposed=SharedWorker]
interface SharedWorkerGlobalScope : WorkerGlobalScope {
  [Replaceable] readonly attribute DOMString name;

  undefined close();

  attribute EventHandler onconnect;
};
```

A SharedWorkerGlobalScope[p1324] object has associated **constructor storage key** (a storage key), **constructor URL** (a URL record), and **credentials** (a credentials mode), and **extended lifetime** (a boolean). They are initialized when the SharedWorkerGlobalScope[p1324] object is created, in the run a worker[p1328] algorithm.

Shared workers receive message ports through connect[p1575] events on their SharedWorkerGlobalScope[p1324] 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** getter steps are to return this's name. Its value represents the name that can be used to obtain a reference to the worker using the SharedWorker constructor.

The **close()** method steps are to close a worker given this.

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:

Event handler	Event handler event type
onconnect	connect

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 Window's 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*'s global object is a WorkerGlobalScope object (i.e., if we are creating a nested dedicated worker), then the relevant owner is that global object. Otherwise, *o*'s global object is a Window object, and the relevant owner is that Window's associated Document.

A user agent has an implementation-defined value, the **between-loads shared worker timeout**, which is some small amount of time. This represents how long the user agent allows shared workers to survive while a page is loading, in case that page is going to contact the shared worker again. Setting this value to greater than zero lets user agents avoid the cost of restarting a shared worker used by a site when the user is navigating from page to page within that site.

Note

A typical value for the between-loads shared worker timeout might be 5 seconds.

A user agent has an implementation-defined value, the **extended lifetime shared worker timeout**, which is some amount of time. This represents how long the user agent allows shared workers which the web developer has requested be given an extended lifetime, to survive and perform work after all of their owners have disappeared. User agents should choose a value that is equal to the

longest lifetime that a service worker can stay alive without any clients, and must not choose a value that is longer than that amount of time.

Note

A typical value for the extended lifetime shared worker timeout[p1325] could be anywhere from 10 seconds to 5 minutes, depending on the implementation's individual policies regarding background work being performed by an origin without a user-visible representation.

A `WorkerGlobalScope`[p1322] *global* is **in extended lifetime** if the following algorithm returns true:

1. If *global* is not a `SharedWorkerGlobalScope`[p1324], then return false.
2. If *global*'s extended lifetime[p1324] is false, then return false.
3. If *global*'s owner set[p1322] is empty, but has been empty for less than the user agent's extended lifetime shared worker timeout[p1325], then return true.
4. For each *owner* of *global*'s owner set[p1322]:
    1. If *owner* is a `Document`[p135] that is fully active[p1051], then return false.

    Note

    In this case global *will be actively needed[p1326], but it's not in extended lifetime.*
5. For each *owner* of *global*'s owner set[p1322]:
    1. If *owner* is a `Document`[p135] and the amount of time that *owner* has been non-fully active[p1051] is less than the user agent's extended lifetime shared worker timeout[p1325], then return true.
    2. If *owner* is a `WorkerGlobalScope`[p1322] that is in extended lifetime[p1326], then return true.
6. Return false.

A `WorkerGlobalScope`[p1322] *global* is **actively needed** if the following algorithm returns true:

1. If *global* is in extended lifetime[p1326], then return true.
2. For each *owner* of *global*'s owner set[p1322]:
    1. If *owner* is a `Document`[p135], and *owner* is fully active[p1051], then return true.
    2. If *owner* is a `WorkerGlobalScope`[p1322] that is actively needed[p1326], then return true.
3. Return false.

A `WorkerGlobalScope`[p1322] *global* is **protected** if the following algorithm returns true:

1. If *global* is not actively needed[p1326], then return false.
2. If *global* is in extended lifetime[p1326], then return true.
3. If *global* is a `SharedWorkerGlobalScope`[p1324], then return true.
4. If *global*'s the worker's ports[p1325] is not empty, then return true.
5. If *global* has outstanding timers, database transactions, or network connections, then return true.
6. Return false.

A `WorkerGlobalScope`[p1322] *global* is **permissible** if the following algorithm returns true:

1. If *global*'s owner set[p1322] is not empty, then return true.
2. If *global* is in extended lifetime[p1326], then return true.
3. If all of the following are true:

- *global* is a SharedWorkerGlobalScope[p1324];
- *global*'s owner set[p1322] has been empty for no more than the user agent's between-loads shared worker timeout[p1325]; and
- the user agent has a navigable[p1035] whose active document[p1036] is not completely loaded[p1114],

then return true.

4. Return false.

¶[p1327] **Note**

The following relationships hold between these terms:

- *Every WorkerGlobalScope[p1322] that is in extended lifetime[p1326] is actively needed[p1326], protected[p1326], and permissible[p1326].*
- *Every WorkerGlobalScope[p1322] that is actively needed[p1326] or protected[p1326] is permissible[p1326].*
- *Every WorkerGlobalScope[p1322] that is protected[p1326] is actively needed[p1326].*

However, the converses do not always hold:

- *A WorkerGlobalScope[p1322] can be actively needed[p1326] but not protected[p1326], if it's a dedicated worker with no outstanding async work that is still performing computation on behalf of a fully active owner, but whose corresponding Worker[p1332] object has been garbage collected. Because of the garbage collection[p1302], the ports[p1325] collection is empty, so it is no longer protected. However, its event loop has not yet yielded to empty its owner set[p1330], so it is still actively needed.*
- *A WorkerGlobalScope[p1322] can be permissible[p1326] but not protected[p1326] or actively needed[p1326], if all the Document[p135]s in its transitive set of owners are in bfcache[p1054], or if it's a SharedWorkerGlobalScope[p1324] with no current owners being kept alive for the duration of the between-loads shared worker timeout[p1325].*

¶[p1327] **Note**

The between-loads shared worker timeout[p1325] only influences the definition of permissible[p1326], not protected[p1326], and so implementations are not required to keep shared workers alive for that duration. Rather, they are required to close shared workers after the timeout is reached.

In contrast, the extended lifetime shared worker timeout[p1325] does influence the definition of protected[p1326], so implementations are required to keep shared workers alive (and not suspended) for that duration.

A WorkerGlobalScope[p1322] *global* is **suspendable** if the following algorithm returns true:

1. If *global* is actively needed[p1326], then return false.
2. If *global* is permissible[p1326], then return true.
3. Return false.

¶[p1327] **Note**

These concepts are used elsewhere in the specification's normative requirements as follows:

- *Workers get closed as orphans[p1329] between when they stop being protected[p1326] and when they stop being permissible[p1326].*
- *Workers that have been closed, but keep executing, can be terminated[p1330] at the user agent's discretion, once they stop being actively needed[p1326].*
- *Workers get suspended or un-suspended[p1329] based on whether they are suspendable[p1327].*

10.2.4 Processing model §p1328

When a user agent is to **run a worker** for a script with `Worker`p1332 or `SharedWorker`p1333 object *worker*, URL *url*, environment settings object[p1144] *outside settings*, `MessagePort`p1299 *outside port*, and a `WorkerOptions`p1332 dictionary *options*, it must run the following steps:

1. Let *is shared* be true if *worker* is a `SharedWorker`p1333 object, and false otherwise.
2. Let *owner* be the relevant owner to add[p1325] given *outside settings*.
3. Let *unsafeWorkerCreationTime* be the unsafe shared current time.
4. Let *agent* be the result of obtaining a dedicated/shared worker agent[p1142] given *outside settings* and *is shared*. Run the rest of these steps in that agent.
5. Let *realm execution context* be the result of creating a new realm[p1145] given *agent* and the following customizations:
    - For the global object, if *is shared* is true, create a new `SharedWorkerGlobalScope`p1324 object. Otherwise, create a new `DedicatedWorkerGlobalScope`p1324 object.
6. Let *worker global scope* be the global object[p1145] of *realm execution context*'s Realm component.

    Note

    This is the `DedicatedWorkerGlobalScope`p1324 or `SharedWorkerGlobalScope`p1324 object created in the previous step.

7. Set up a worker environment settings object[p1331] with *realm execution context*, *outside settings*, and *unsafeWorkerCreationTime*, and let *inside settings* be the result.
8. Set *worker global scope*'s name[p1323] to *options*["`name`p1332"].
9. Append *owner* to *worker global scope*'s owner set[p1322].
10. If *is shared* is true:
    1. Set *worker global scope*'s constructor storage key[p1324] to the result of running obtain a storage key for non-storage purposes given *outside settings*.
    2. Set *worker global scope*'s constructor URL[p1324] to *url*.
    3. Set *worker global scope*'s type[p1323] to *options*["`type`p1332"].
    4. Set *worker global scope*'s credentials[p1324] to *options*["`credentials`p1332"].
    5. Set *worker global scope*'s extended lifetime[p1324] to *options*["`extendedLifetime`p1333"].
11. Let *destination* be "`sharedworker`" if *is shared* is true, and "`worker`" otherwise.
12. Obtain *script* by switching on *options*["`type`p1332"]:

    ↪ "`classic`"

    Fetch a classic worker script[p1156] given *url*, *outside settings*, *destination*, *inside settings*, and with *onComplete* and *performFetch* as defined below.

    ↪ "`module`"

    Fetch a module worker script graph[p1158] given *url*, *outside settings*, *destination*, the value of the `credentials` member of *options*, *inside settings*, and with *onComplete* and *performFetch* as defined below.

    In both cases, let *performFetch* be the following perform the fetch hook[p1155] given *request*, *isTopLevel*[p1155], and *processCustomFetchResponse*[p1155]:

    1. If *isTopLevel* is false, fetch *request* with *processResponseConsumeBody* set to *processCustomFetchResponse*, and abort these steps.
    2. Set *request*'s reserved client to *inside settings*.
    3. Fetch *request* with *processResponseConsumeBody* set to the following steps given response *response* and null, failure, or a byte sequence *bodyBytes*:
        1. Set *worker global scope*'s url[p1323] to *response*'s URL.

2. Set *inside settings*'s creation URL[p1143] to *response*'s URL.

3. Initialize worker global scope's policy container[p961] given *worker global scope*, *response*, and *inside settings*.

4. If the Run CSP initialization for a global object algorithm returns "`Blocked`" when executed upon *worker global scope*, set *response* to a network error. [CSP][p1580]

5. If *worker global scope*'s embedder policy[p1323]'s value[p955] is compatible with cross-origin isolation[p955] and *is shared* is true, then set *agent*'s agent cluster's cross-origin isolation mode[p1141] to "`logical`[p1050]" or "`concrete`[p1050]". The one chosen is implementation-defined.

   This really ought to be set when the agent cluster is created, which requires a redesign of this section.

6. If the result of checking a global object's embedder policy[p956] with *worker global scope*, *outside settings*, and *response* is false, then set *response* to a network error.

7. Set *worker global scope*'s cross-origin isolated capability[p1323] to true if *agent*'s agent cluster's cross-origin isolation mode[p1141] is "`concrete`[p1050]".

8. If *is shared* is false and *owner*'s cross-origin isolated capability[p1144] is false, then set *worker global scope*'s cross-origin isolated capability[p1323] to false.

9. If *is shared* is false and *response*'s URL's scheme is "`data`", then set *worker global scope*'s cross-origin isolated capability[p1323] to false.

   Note

   This is a conservative default for now, while we figure out how workers in general, and `data:` URL workers in particular (which are cross-origin from their owner), will be treated in the context of permissions policies. See w3c/webappsec-permissions-policy issue #207 for more details.

10. Run *processCustomFetchResponse* with *response* and *bodyBytes*.

In both cases, let *onComplete* given *script* be the following steps:

1. If *script* is null or if *script*'s error to rethrow[p1153] is non-null:

   1. Queue a global task[p1195] on the DOM manipulation task source[p1203] given *worker*'s relevant global object[p1151] to fire an event named `error`[p1575] at *worker*.

   2. Run the environment discarding steps[p1144] for *inside settings*.

   3. Abort these steps.

2. Associate *worker* with *worker global scope*.

3. Let *inside port* be a new `MessagePort`[p1299] object in *inside settings*'s realm[p1145].

4. If *is shared* is false:

   1. Set *inside port*'s message event target[p1300] to *worker global scope*.

   2. Set *worker global scope*'s inside port[p1324] to *inside port*.

5. Entangle[p1300] *outside port* and *inside port*.

6. Create a new `WorkerLocation`[p1337] object and associate it with *worker global scope*.

7. *Closing orphan workers*: Start monitoring *worker global scope* such that no sooner than it stops being protected[p1326], and no later than it stops being permissible[p1326], *worker global scope*'s closing[p1325] flag is set to true.

8. *Suspending workers*: Start monitoring *worker global scope*, such that whenever *worker global scope*'s closing[p1325] flag is false and it is suspendable[p1327], the user agent suspends execution of script in *worker global scope* until such time as either the closing[p1325] flag switches to true or *worker global scope* stops being suspendable[p1327].

9. Set *inside settings*'s execution ready flag[p1144].

10. If *script* is a classic script[p1153], then run the classic script[p1165] *script*. Otherwise, it is a module script[p1153]; run the

module script[p1165] *script*.

Note

In addition to the usual possibilities of returning a value or failing due to an exception, this could be prematurely aborted[p1166] by the terminate a worker[p1330] algorithm defined below.

11. Enable *outside port*'s port message queue[p1300].
12. If *is shared* is false, enable the port message queue[p1300] of the worker's implicit port.
13. If *is shared* is true, then queue a global task[p1195] on the DOM manipulation task source[p1203] given *worker global scope* to fire an event named `connect`[p1575] at *worker global scope*, using `MessageEvent`[p1283], with the `data`[p1284] attribute initialized to the empty string, the `ports`[p1284] attribute initialized to a new frozen array containing *inside port*, and the `source`[p1284] attribute initialized to *inside port*.
14. Enable the client message queue of the `ServiceWorkerContainer` object whose associated service worker client is *worker global scope*'s relevant settings object[p1151].
15. *Event loop*: Run the responsible event loop[p1145] specified by *inside settings* until it is destroyed.

    Note

    The handling of events or the execution of callbacks by tasks[p1194] run by the event loop[p1193] might get prematurely aborted[p1166] by the terminate a worker[p1330] algorithm defined below.

    Note

    The worker processing model remains on this step until the event loop is destroyed, which happens after the closing[p1325] flag is set to true, as described in the event loop[p1193] processing model.
16. Clear the *worker global scope*'s map of active timers[p1256].
17. Disentangle all the ports in the list of the worker's ports[p1325].
18. Empty *worker global scope*'s owner set[p1322].

When a user agent is to **terminate a worker**, it must run the following steps in parallel[p44] with the worker's main loop (the "run a worker[p1328]" processing model defined above):

1. Set the worker's `WorkerGlobalScope`[p1322] object's closing[p1325] flag to true.
2. If there are any tasks[p1194] queued in the `WorkerGlobalScope`[p1322] object's relevant agent[p1141]'s event loop[p1193]'s task queues[p1193], discard them without processing them.
3. Abort the script[p1166] currently running in the worker.
4. If the worker's `WorkerGlobalScope`[p1322] object is actually a `DedicatedWorkerGlobalScope`[p1324] object (i.e. the worker is a dedicated worker), then empty the port message queue[p1300] of the port that the worker's implicit port is entangled with.

User agents may invoke the terminate a worker[p1330] algorithm when a worker stops being actively needed[p1326] and the worker continues executing even after its closing[p1325] flag was set to true.

10.2.5 Runtime script errors §[p1330]

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 will report[p1167] it for the worker's `WorkerGlobalScope`[p1322] object.

10.2.6 Creating workers §p1331

10.2.6.1 The AbstractWorker p1331 mixin §p1331

IDL

```
interface mixin AbstractWorker {
  attribute EventHandler onerror;
};
```

The following are the event handlers p1206 (and their corresponding event handler event types p1209) that must be supported, as event handler IDL attributes p1207, by objects implementing the AbstractWorker p1331 interface:

Event handler p1206	Event handler event type p1209
onerror	error p1575

10.2.6.2 Script settings for workers §p1331

To **set up a worker environment settings object**, given a JavaScript execution context *execution context*, an environment settings object p1144 *outside settings*, and a number *unsafeWorkerCreationTime*:

1. Let *realm* be the value of *execution context*'s Realm component.
2. Let *worker global scope* be *realm*'s global object p1145.
3. Let *origin* be a unique opaque origin p939 if *worker global scope*'s url p1323's scheme is "data"; otherwise *outside settings*'s origin p1144.
4. Let *settings object* be a new environment settings object p1144 whose algorithms are defined as follows:

   The realm execution context p1144

   Return *execution context*.

   The module map p1144

   Return *worker global scope*'s module map p1323.

   The API base URL p1144

   Return *worker global scope*'s url p1323.

   The origin p1144

   Return *origin*.

   The has cross-site ancestor p1144

   1. If *outside settings*'s has cross-site ancestor p1144 is true, then return true.
   2. If *worker global scope*'s url p1323's scheme is "data", then return true.
   3. Return false.

   The policy container p1144

   Return *worker global scope*'s policy container p1323.

   The cross-origin isolated capability p1144

   Return *worker global scope*'s cross-origin isolated capability p1323.

   The time origin p1145

   Return the result of coarsening *unsafeWorkerCreationTime* with *worker global scope*'s cross-origin isolated capability p1323.

5. Set *settings object*'s id p1143 to a new unique opaque string, creation URL p1143 to *worker global scope*'s url p1323, top-level creation URL p1144 to null, target browsing context p1144 to null, and active service worker p1144 to null.
6. If *worker global scope* is a DedicatedWorkerGlobalScope p1324 object, then set *settings object*'s top-level origin p1144 to *outside settings*'s top-level origin p1144.

7. Otherwise, set *settings object*'s top-level origin[p1144] 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*.

10.2.6.3 Dedicated workers and the `Worker`[p1332] interface §[p1332]

IDL

```
[Exposed=(Window,DedicatedWorker,SharedWorker)]
interface Worker : EventTarget {
  constructor((TrustedScriptURL or USVString) scriptURL, optional WorkerOptions options = {});

  undefined terminate();

  undefined postMessage(any message, sequence<object> transfer);
  undefined postMessage(any message, optional StructuredSerializeOptions options = {});
};

dictionary WorkerOptions {
  DOMString name = "";
  WorkerType type = "classic";
  RequestCredentials credentials = "same-origin"; // credentials is only used if type is "module"
};

enum WorkerType { "classic", "module" };

Worker includes AbstractWorker;
Worker includes MessageEventTarget;
```

For web developers (non-normative)

***worker* = new `Worker`[p1333](*scriptURL* [, *options*])**

Returns a new `Worker`[p1332] object. *scriptURL* will be fetched and executed in the background, creating a new global environment for which *worker* represents the communication channel.

options can contain the following values:

- `name`[p1332] can be used to define the name[p1323] of that global environment, primarily for debugging purposes.
- `type`[p1332] can be used to load the new global environment from *scriptURL* as a JavaScript module, by setting it to the value "`module`".
- `credentials`[p1332] can be used to specify how *scriptURL* is fetched, but only if `type`[p1332] is set to "`module`".

***worker*.`terminate`[p1332]()**

Aborts *worker*'s associated global environment.

***worker*.`postMessage`[p1332](*message* [, *transfer*])**

***worker*.`postMessage`[p1332](*message* [, { `transfer`[p1299] }])**

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.

Each `Worker`[p1332] object has an associated **outside port** (a `MessagePort`[p1299]). 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`[p1332] object.

The **`terminate()`** method steps are to terminate a worker[p1330] given this's worker.

The **`postMessage(message, transfer)`** and **`postMessage(message, options)`** methods on `Worker`[p1332] objects act as if, when invoked, they immediately invoked the respective `postMessage(message, transfer)`[p1302] and `postMessage(message, options)`[p1302] on this's outside port[p1332], with the same arguments, and returned the same return value.

Example

The postMessage()[p1332] method's first argument can be structured data:

```
worker.postMessage({opcode: 'activate', device: 1938, parameters: [23, 102]});
```

The **new Worker(*scriptURL*, *options*)** constructor steps are:

1. Let *compliantScriptURL* be the result of invoking the get trusted type compliant string algorithm with TrustedScriptURL, this's relevant global object[p1151], *scriptURL*, "Worker constructor", and "script".
2. Let *outsideSettings* be this's relevant settings object[p1151].
3. Let *workerURL* be the result of encoding-parsing a URL[p101] given *compliantScriptURL*, relative to *outsideSettings*.

   Note

   Any same-origin[p940] URL (including blob: URLs) can be used. data: URLs can also be used, but they create a worker with an opaque origin[p939].
4. If *workerURL* is failure, then throw a "SyntaxError" DOMException.
5. Let *outsidePort* be a new MessagePort[p1299] in *outsideSettings*'s realm[p1145].
6. Set *outsidePort*'s message event target[p1300] to this.
7. Set this's outside port[p1332] to *outsidePort*.
8. Let *worker* be this.
9. Run this step in parallel[p44]:
   1. Run a worker[p1328] given *worker*, *workerURL*, *outsideSettings*, *outsidePort*, and *options*.

✔ MDN

10.2.6.4 Shared workers and the SharedWorker[p1333] interface §[p1333]

IDL

```
[Exposed=Window]
interface SharedWorker : EventTarget {
  constructor((TrustedScriptURL or USVString) scriptURL, optional (DOMString or SharedWorkerOptions)
options = {});

  readonly attribute MessagePort port;
};
SharedWorker includes AbstractWorker;

dictionary SharedWorkerOptions : WorkerOptions {
  boolean extendedLifetime = false;
};
```

For web developers (non-normative)

***sharedWorker* = new SharedWorker[p1334](*scriptURL* [, *name*])**

Returns a new SharedWorker[p1333] 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[p1323] of that global environment.

***sharedWorker* = new SharedWorker[p1334](*scriptURL* [, *options*])**

Returns a new SharedWorker[p1333] object. *scriptURL* will be fetched and executed in the background, creating a new global environment for which *sharedWorker* represents the communication channel.

options can contain the following values:

- name[p1332] can be used to define the name[p1323] of that global environment.

- `type`[p1332] can be used to load the new global environment from *scriptURL* as a JavaScript module, by setting it to the value "`module`".
- `credentials`[p1332] can be used to specify how *scriptURL* is fetched, but only if `type`[p1332] is set to "`module`".
- `extendedLifetime`[p1333] can be set to request that the newly-created global environment be given extra time to perform its operations even after all of its owners[p1322] have disappeared. This can be useful, for example, for performing asynchronous work after page unload.

⚠Warning!

Note that attempting to construct a shared worker with* options *whose* `type`[p1332], `credentials`[p1332], *or* `extendedLifetime`[p1333] *values mismatch those of an existing shared worker with the same constructor URL*[p1324] *and name*[p1323]*, will cause the returned* sharedWorker *to fire an* `error`[p1575] *event and not connect to the existing shared worker.

***sharedWorker*.port**[p1334]

Returns *sharedWorker*'s `MessagePort`[p1299] object which can be used to communicate with the global environment.

A user agent has an associated **shared worker manager** which is the result of starting a new parallel queue[p44].

Note

Each user agent has a single shared worker manager[p1334] *for simplicity. Implementations could use one per origin*[p939]*; that would not be observably different and enables more concurrency.*

Each `SharedWorker`[p1333] has a **port**, a `MessagePort`[p1299] set when the object is created.

The **`port`** getter steps are to return this's port[p1334].

The **`new SharedWorker(scriptURL, options)`** constructor steps are:

1. Let *compliantScriptURL* be the result of invoking the get trusted type compliant string algorithm with `TrustedScriptURL`, this's relevant global object[p1151], *scriptURL*, "`SharedWorker constructor`", and "`script`".
2. If *options* is a `DOMString`, set *options* to a new `WorkerOptions`[p1332] dictionary whose `name`[p1332] member is set to the value of *options* and whose other members are set to their default values.
3. Let *outsideSettings* be this's relevant settings object[p1151].
4. Let *urlRecord* be the result of encoding-parsing a URL[p101] given *compliantScriptURL*, relative to *outsideSettings*.

   Note

   Any same-origin[p940] *URL (including* `blob:` *URLs) can be used.* `data:` *URLs can also be used, but they create a worker with an opaque origin*[p939]*.*
5. If *urlRecord* is failure, then throw a `"SyntaxError"` `DOMException`.
6. Let *outsidePort* be a new `MessagePort`[p1299] in *outsideSettings*'s realm[p1145].
7. Set this's port[p1334] to *outsidePort*.
8. Let *callerIsSecureContext* be true if *outsideSettings* is a secure context[p1152]; otherwise, false.
9. Let *outsideStorageKey* be the result of running obtain a storage key for non-storage purposes given *outsideSettings*.
10. Let *worker* be this.
11. Enqueue the following steps[p44] to the shared worker manager[p1334]:
    1. Let *workerGlobalScope* be null.
    2. For each *scope* in the list of all `SharedWorkerGlobalScope`[p1324] objects:
       1. If all of the following are true:

- *scope*'s constructor storage key[p1324] equals *outsideStorageKey*;
- *scope*'s closing[p1325] flag is false;
- *scope*'s constructor URL[p1324] equals *urlRecord*; and
- *scope*'s name[p1323] equals *options*["name[p1332]"],

then:

1. Set *workerGlobalScope* to *scope*.
2. Break.

Note

data: URLs create a worker with an opaque origin[p939], which would prevent such a worker from ever being matched if its own storage key were used. Instead, the constructor storage key[p1324] (derived from the creating environment) and constructor URL[p1324] are compared, so the same data: URL can be used within a storage key to get to the same SharedWorkerGlobalScope[p1324] object, but cannot be used to bypass the storage key restriction.

3. If *workerGlobalScope* is not null, but the user agent has been configured to disallow communication between the worker represented by the *workerGlobalScope* and the scripts[p1152] whose settings object[p1152] is *outsideSettings*, then set *workerGlobalScope* to null.

Note

For example, a user agent could have a development mode that isolates a particular top-level traversable[p1037] 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 *workerGlobalScope* is not null, and any of the following are true:
   - *workerGlobalScope*'s type[p1323] is not equal to *options*["type[p1332]"];
   - *workerGlobalScope*'s credentials[p1324] is not equal to *options*["credentials[p1332]"]; or
   - *workerGlobalScope*'s extended lifetime[p1324] is not equal to *options*["extendedLifetime[p1333]"],

   then:
   1. Queue a global task[p1195] on the DOM manipulation task source[p1203] given *worker*'s relevant global object[p1151] to fire an event named error[p1575] at *worker*.
   2. Abort these steps.
5. If *workerGlobalScope* is not null:
   1. Let *insideSettings* be *workerGlobalScope*'s relevant settings object[p1151].
   2. Let *workerIsSecureContext* be true if *insideSettings* is a secure context[p1152]; otherwise, false.
   3. If *workerIsSecureContext* is not *callerIsSecureContext*:
      1. Queue a global task[p1195] on the DOM manipulation task source[p1203] given *worker*'s relevant global object[p1151] to fire an event named error[p1575] at *worker*.
      2. Abort these steps.
   4. Associate *worker* with *workerGlobalScope*.
   5. Let *insidePort* be a new MessagePort[p1299] in *insideSettings*'s realm[p1145].
   6. Entangle[p1300] *outsidePort* and *insidePort*.
   7. Queue a global task[p1195] on the DOM manipulation task source[p1203] given *workerGlobalScope* to fire an event named connect[p1575] at *workerGlobalScope*, using MessageEvent[p1283], with the data[p1284] attribute initialized to the empty string, the ports[p1284] attribute initialized to a new frozen array containing only *insidePort*, and the source[p1284] attribute initialized to *insidePort*.

8. Append the relevant owner to add[p1325] given *outsideSettings* to *workerGlobalScope*'s owner set[p1322].

6. Otherwise, in parallel[p44], run a worker[p1328] given *worker*, *urlRecord*, *outsideSettings*, *outsidePort*, and *options*.

10.2.7 Concurrent hardware capabilities §[p13 36]

```
IDL
interface mixin NavigatorConcurrentHardware {
  readonly attribute unsigned long long hardwareConcurrency;
};
```

For web developers (non-normative)

***self*.navigator[p1262].hardwareConcurrency[p1336]**

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[p1332] that can be created) or when the user agent desires to limit fingerprinting possibilities.

10.3 APIs available to workers §[p13 36]

10.3.1 Importing scripts and libraries §[p13 36]

The **importScripts(...*urls*)** method steps are:

1. Let *urlStrings* be « ».
2. For each *url* of *urls*:
    1. Append the result of invoking the get trusted type compliant string algorithm with TrustedScriptURL, this's relevant global object[p1151], *url*, "`WorkerGlobalScope importScripts`", and "`script`" to *urlStrings*.
3. Import scripts into worker global scope[p1336] given this and *urlStrings*.

To **import scripts into worker global scope**, given a WorkerGlobalScope[p1322] object *worker global scope*, a list of scalar value strings *urls*, and an optional perform the fetch hook[p1155] *performFetch*:

1. If *worker global scope*'s type[p1323] is "`module`", throw a TypeError exception.
2. Let *settings object* be the current settings object[p1151].
3. If *urls* is empty, return.
4. Let *urlRecords* be « ».
5. For each *url* of *urls*:
    1. Let *urlRecord* be the result of encoding-parsing a URL[p101] given *url*, relative to *settings object*.
    2. If *urlRecord* is failure, then throw a "SyntaxError" DOMException.
    3. Append *urlRecord* to *urlRecords*.
6. For each *urlRecord* of *urlRecords*:
    1. Fetch a classic worker-imported script[p1157] given *urlRecord* and *settings object*, passing along *performFetch* if provided. If this succeeds, let *script* be the result. Otherwise, rethrow the exception.
    2. Run the classic script[p1165] *script*, with *rethrow errors* set to true.

Note

script *will run until it either returns, fails to parse, fails to catch an exception, or gets prematurely aborted[p1166] by the terminate a worker[p1330] algorithm defined above.*

If an exception was thrown or if the script was prematurely aborted[p1166], then abort all these steps, letting the exception or aborting continue to be processed by the calling script[p1152].

Note

Service Workers *is an example of a specification that runs this algorithm with its own perform the fetch hook[p1155]. [SW][p1587]*

10.3.2 The WorkerNavigator[p1337] interface §[p13 37]

The **navigator** attribute of the WorkerGlobalScope[p1322] interface must return an instance of the WorkerNavigator[p1337] interface, which represents the identity and state of the user agent (the client):

IDL

```
[Exposed=Worker]
interface WorkerNavigator {};
WorkerNavigator includes NavigatorID;
WorkerNavigator includes NavigatorLanguage;
WorkerNavigator includes NavigatorOnLine;
WorkerNavigator includes NavigatorConcurrentHardware;
```

10.3.3 The WorkerLocation[p1337] interface §[p13 37]

IDL

```
[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[p1337] object has an associated **WorkerGlobalScope object** (a WorkerGlobalScope[p1322] object).

The **href** getter steps are to return this's WorkerGlobalScope object[p1337]'s url[p1323], serialized.

The **origin** getter steps are to return the serialization[p939] of this's WorkerGlobalScope object[p1337]'s url[p1323]'s origin.

The **protocol** getter steps are to return this's WorkerGlobalScope object[p1337]'s url[p1323]'s scheme, followed by ":".

The **host** getter steps are:

1. Let *url* be this's WorkerGlobalScope object[p1337]'s url[p1323].
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** getter steps are:

1. Let *host* be this's WorkerGlobalScope object[p1337]'s url[p1323]'s host.

2. If *host* is null, return the empty string.
3. Return *host*, serialized.

The **port** getter steps are:

1. Let *port* be this's WorkerGlobalScope object[p1337]'s url[p1323]'s port.
2. If *port* is null, return the empty string.
3. Return *port*, serialized.

The **pathname** getter steps are to return the result of URL path serializing this's WorkerGlobalScope object[p1337]'s url[p1323].

The **search** getter steps are:

1. Let *query* be this's WorkerGlobalScope object[p1337]'s url[p1323]'s query.
2. If *query* is either null or the empty string, return the empty string.
3. Return "?", followed by *query*.

The **hash** getter steps are:

1. Let *fragment* be this's WorkerGlobalScope object[p1337]'s url[p1323]'s fragment.
2. If *fragment* is either null or the empty string, return the empty string.
3. Return "#", followed by *fragment*.

11 Worklets §[p13 39]

11.1 Introduction §[p13 39]

This section is non-normative.

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.

11.1.1 Motivations §[p13 39]

This section is non-normative.

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`[p965], 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`[p965] environment would restrict user agents to performing work in the same thread as the `Window`[p965] 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`[p1342].

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[p1145] 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`[p1342] is expected to be shared between multiple separate scripts. (This is similar to how a single `Window`[p965] 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. [CSSPAINT][p1581] [WEBAUDIO][p1587]

11.1.2 Code idempotence §[p13 39]

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`[p1342]s 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`[p1323] on `WorkletGlobalScope`[p1342]).

  > 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[p1153], 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[p1145].

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`[p1342] instances per `Worklet`[p1345], and randomly assign a method or set of methods on a class to a particular `WorkletGlobalScope`[p1342] 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`[p1342] subclasses at any time.

11.1.3 Speculative evaluation §[p13 40]

Some specifications which use worklets can invoke methods on a web-developer-provided class based on the state of the user agent. To increase concurrency between threads, a user agent may invoke a method speculatively, based on potential future states.

In these specifications, user agents might invoke such methods at any time, and with any arguments, not just ones corresponding to the current state of the user agent. The results of such speculative evaluations are not displayed immediately, but can be cached for use if the user agent state matches the speculated state. This can increase the concurrency between the user agent and worklet threads.

As a result of this, to prevent interoperability risks between user agents, authors who register classes on such `WorkletGlobalScope`[p1342]s should make their code stateless. That is, the only effect of invoking a method should be its result, and not any side effects such as updating mutable state.

The same techniques which encourage code idempotence[p1339] also encourage authors to write stateless code.

11.2 Examples §[p13 40]

This section is non-normative.

For these examples, we'll use a fake worklet. The `Window`[p965] object provides two `Worklet`[p1345] instances, which each run code in their own collection of `FakeWorkletGlobalScope`[p1341]s:

IDL
```
partial interface Window {
  [SameObject, SecureContext] readonly attribute Worklet fakeWorklet1;
  [SameObject, SecureContext] readonly attribute Worklet fakeWorklet2;
};
```

Each `Window`[p965] has two `Worklet`[p1345] instances, **fake worklet 1** and **fake worklet 2**. Both of these have their worklet global scope type[p1345] set to `FakeWorkletGlobalScope`[p1341], and their worklet destination type[p1345] set to "`fakeworklet`". User agents should create at least two `FakeWorkletGlobalScope`[p1341] instances per worklet.

Note

"`fakeworklet`" *is not actually a valid destination per* Fetch. *But this illustrates how real worklets would generally have their own worklet-type-specific destination.* *[FETCH]*[p1582]

The **fakeWorklet1** getter steps are to return this's fake worklet 1[p1340].

The **fakeWorklet2** getter steps are to return this's fake worklet 2[p1340].

IDL
```
[Global=(Worklet,FakeWorklet),
 Exposed=FakeWorklet,
 SecureContext]
interface FakeWorkletGlobalScope : WorkletGlobalScope {
  undefined registerFake(DOMString type, Function classConstructor);
};
```

Each FakeWorkletGlobalScope[p1341] 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[p1341][*type*] to *classConstructor*.

11.2.1 Loading scripts §[p13 41]

This section is non-normative.

To load scripts into fake worklet 1[p1340], a web developer would write:

```
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[p1340].

If a web developer wants to perform a task only after the scripts have successfully run and loaded into some worklets, they could write:

```
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[p1342]s per Worklet[p1345], as discussed in the section on code idempotence[p1339]. In particular, the specification above for fake worklet 1[p1340] and fake worklet 2[p1340] require this. So, consider a scenario such as the following:

```
// 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!
[fakeWorklet1#2] Hello from a FakeWorkletGlobalScope!
[fakeWorklet1#3] Hello from a FakeWorkletGlobalScope!
```

If the user agent at some point decided to kill and restart the third instance of FakeWorkletGlobalScope[p1341], the console would again print `[fakeWorklet1#3] Hello from a FakeWorkletGlobalScope!` when this occurs.

11.2.2 Registering a class and invoking its methods §[p13 42]

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:

```
// 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[p1345] *worklet*:

1. Optionally, create a worklet global scope[p1343] for *worklet*.
2. Let *workletGlobalScope* be one of *worklet*'s global scopes[p1345], chosen in an implementation-defined manner.
3. Let *classConstructor* be *workletGlobalScope*'s registered class constructors map[p1341]["`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 « true » and "`rethrow`", and with *callback this value* set to *classInstance*.

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()`[p1341] method steps.

11.3 Infrastructure §[p13 42]

11.3.1 The global scope §[p13 42]

Subclasses of WorkletGlobalScope[p1342] are used to create global objects[p1145] wherein code loaded into a particular Worklet[p1345] can execute.

IDL

```
[Exposed=Worklet, SecureContext]
interface WorkletGlobalScope {};
```

Note

Other specifications are intended to subclass WorkletGlobalScope[p1342], adding APIs to register a class, as well as other APIs specific for their worklet type.

Each WorkletGlobalScope[p1342] has an associated **module map**. It is a module map[p1189], initially empty.

11.3.1.1 Agents and event loops §p1343

This section is non-normative.

Each WorkletGlobalScope[p1342] is contained in its own worklet agent[p1140], which has its corresponding event loop[p1193]. However, in practice, implementation of these agents and event loops is expected to be different from most others.

A worklet agent[p1140] exists for each WorkletGlobalScope[p1342] since, in theory, an implementation could use a separate thread for each WorkletGlobalScope[p1342] 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[p1140] ("the main thread"). Contrast this with dedicated worker agents[p1140], whose true value for [[CanBlock]] effectively requires them to get a dedicated operating system thread.

Worklet event loops[p1193] are also somewhat special. They are only used for tasks[p1194] associated with addModule()[p1346], tasks wherein the user agent invokes author-defined methods, and microtasks[p1194]. Thus, even though the event loop processing model[p1196] 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[p1200].

11.3.1.2 Creation and termination §p1343

To **create a worklet global scope** for a Worklet[p1345] *worklet*:

1. Let *outsideSettings* be *worklet*'s relevant settings object[p1151].
2. Let *agent* be the result of obtaining a worklet agent[p1142] given *outsideSettings*. Run the rest of these steps in that agent.
3. Let *realmExecutionContext* be the result of creating a new realm[p1145] 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[p1345].
4. Let *workletGlobalScope* be the global object[p1145] of *realmExecutionContext*'s Realm component.
5. Let *insideSettings* be the result of setting up a worklet environment settings object[p1344] given *realmExecutionContext* and *outsideSettings*.
6. Let *pendingAddedModules* be a clone of *worklet*'s added modules list[p1345].
7. Let *runNextAddedModule* be the following steps:
    1. If *pendingAddedModules* is not empty:
        1. Let *moduleURL* be the result of dequeuing from *pendingAddedModules*.
        2. Fetch a worklet script graph[p1158] given *moduleURL*, *insideSettings*, *worklet*'s worklet destination type[p1345], what credentials mode?, *insideSettings*, *worklet*'s module responses map[p1345], and with the following steps given *script*:

            Note

            This will not actually perform a network request, as it will just reuse responses from worklet's *module responses map[p1345]. The main purpose of this step is to create a new* workletGlobalScope*-specific module script[p1153] from the response.*

            1. Assert: *script* is not null, since the fetch succeeded and the source text was successfully parsed when *worklet*'s module responses map[p1345] was initially populated with *moduleURL*.
            2. Run a module script[p1165] given *script*.
            3. Run *runNextAddedModule*.
        3. Abort these steps.
    2. Append *workletGlobalScope* to *outsideSettings*'s global object[p1145]'s associated Document[p966]'s worklet global scopes[p1347].
    3. Append *workletGlobalScope* to *worklet*'s global scopes[p1345].

4. Run the responsible event loop[p1145] specified by *insideSettings*.

8. Run *runNextAddedModule*.

To **terminate a worklet global scope** given a `WorkletGlobalScope`[p1342] *workletGlobalScope*:

1. Let *eventLoop* be *workletGlobalScope*'s relevant agent[p1141]'s event loop[p1193].
2. If there are any tasks[p1194] queued in *eventLoop*'s task queues[p1193], discard them without processing them.
3. Wait for *eventLoop* to complete the currently running task[p1194].
4. If the previous step doesn't complete within an implementation-defined period of time, then abort the script[p1166] currently running in the worklet.
5. Destroy *eventLoop*.
6. Remove *workletGlobalScope* from the global scopes[p1345] of the `Worklet`[p1345] whose global scopes[p1345] contains *workletGlobalScope*.
7. Remove *workletGlobalScope* from the worklet global scopes[p1347] of the `Document`[p135] whose worklet global scopes[p1347] contains *workletGlobalScope*.

11.3.1.3 Script settings for worklets §[p1344]

To **set up a worklet environment settings object**, given a JavaScript execution context *executionContext* and an environment settings object[p1144] *outsideSettings*:

1. Let *origin* be a unique opaque origin[p939].
2. Let *inheritedAPIBaseURL* be *outsideSettings*'s API base URL[p1144].
3. Let *inheritedPolicyContainer* be a clone[p960] of *outsideSettings*'s policy container[p1144].
4. Let *realm* be the value of *executionContext*'s Realm component.
5. Let *workletGlobalScope* be *realm*'s global object[p1145].
6. Let *settingsObject* be a new environment settings object[p1144] whose algorithms are defined as follows:

   The realm execution context[p1144]

   Return *executionContext*.

   The module map[p1144]

   Return *workletGlobalScope*'s module map[p1342].

   The API base URL[p1144]

   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()`[p1346]*. So this API base URL*[p1144] *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*[p1144]*.*

   The origin[p1144]

   Return *origin*.

   The has cross-site ancestor[p1144]

   Return true.

   The policy container[p1144]

   Return *inheritedPolicyContainer*.

The cross-origin isolated capability[p1144]

Return TODO.

The time origin[p1145]

Assert: this algorithm is never called, because the time origin[p1145] is not available in a worklet context.

7. Set *settingsObject*'s id[p1143] to a new unique opaque string, creation URL[p1143] to *inheritedAPIBaseURL*, top-level creation URL[p1144] to null, top-level origin[p1144] to *outsideSettings*'s top-level origin[p1144], target browsing context[p1144] to null, and active service worker[p1144] to null.
8. Set *realm*'s [[HostDefined]] field to *settingsObject*.
9. Return *settingsObject*.

11.3.2 The Worklet[p1345] class §[p1345]

The Worklet[p1345] class provides the capability to add module scripts into its associated WorkletGlobalScope[p1342]s. The user agent can then create classes registered on the WorkletGlobalScope[p1342]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[p1345] 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[p1342]; and
- its **worklet destination type**, which must be a destination, and is used when fetching scripts.

For web developers (non-normative)

await *worklet*.addModule[p1346](*moduleURL*[, { credentials[p1345] }])

Loads and executes the module script[p1153] given by *moduleURL* into all of *worklet*'s global scopes[p1345]. 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[p1345] option can be set to a credentials mode to modify the script-fetching process. It defaults to "same-origin".

Any failures in fetching[p1158] 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[p1345] has a list of **global scopes**, which contains instances of the Worklet[p1345]'s worklet global scope type[p1345]. It is initially empty.

A Worklet[p1345] has an **added modules list**, which is a list of URLs, initially empty. Access to this list should be thread-safe.

A Worklet[p1345] has a **module responses map**, which is an ordered map from URLs to either "fetching" or tuples consisting of a response and either null, failure, or a byte sequence representing the response body. This map is initially empty, and access to it should be thread-safe.

Note

The added modules list[p1345] and module responses map[p1345] exist to ensure that WorkletGlobalScope[p1342]s created at different times get equivalent module scripts[p1153] run in them, based on the same source text. This allows the creation of additional WorkletGlobalScope[p1342]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()`[p1346] 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[p1345]) to each thread which has a `WorkletGlobalScope`[p1342].

Then, when a user agent creates[p1343] a new `WorkletGlobalScope`[p1342] for a given `Worklet`[p1345], 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`[p1342].

The **`addModule(moduleURL, options)`** method steps are:

1. Let *outsideSettings* be the relevant settings object[p1151] of this.
2. Let *moduleURLRecord* be the result of encoding-parsing a URL[p101] given *moduleURL*, relative to *outsideSettings*.
3. If *moduleURLRecord* is failure, then return a promise rejected with a `"SyntaxError"` `DOMException`.
4. Let *promise* be a new promise.
5. Let *workletInstance* be this.
6. Run the following steps in parallel[p44]:
    1. If *workletInstance*'s global scopes[p1345] is empty:
        1. Create a worklet global scope[p1343] given *workletInstance*.
        2. Optionally, create[p1343] additional global scope instances given *workletInstance*, depending on the specific worklet in question and its specification.
        3. Wait for all steps of the creation[p1343] process(es) — including those taking place within the worklet agents[p1140] — to complete, before moving on.
    2. Let *pendingTasks* be *workletInstance*'s global scopes[p1345]'s size.
    3. Let *addedSuccessfully* be false.
    4. For each *workletGlobalScope* of *workletInstance*'s global scopes[p1345], queue a global task[p1195] on the networking task source[p1204] given *workletGlobalScope* to fetch a worklet script graph[p1158] given *moduleURLRecord*, *outsideSettings*, *workletInstance*'s worklet destination type[p1345], *options*["`credentials`[p1345]"], *workletGlobalScope*'s relevant settings object[p1151], *workletInstance*'s module responses map[p1345], and the following steps given *script*:

        Note

        Only the first of these fetches will actually perform a network request; the ones for other `WorkletGlobalScope`[p1342]s will reuse responses from workletInstance's *module responses map[p1345].*

        1. If *script* is null:
            1. Queue a global task[p1195] on the networking task source[p1204] given *workletInstance*'s relevant global object[p1151] to perform the following steps:
                1. If *pendingTasks* is not −1:
                    1. Set *pendingTasks* to −1.
                    2. Reject *promise* with an `"AbortError"` `DOMException`.
            2. Abort these steps.
        2. If *script*'s error to rethrow[p1153] is not null:
            1. Queue a global task[p1195] on the networking task source[p1204] given *workletInstance*'s relevant global object[p1151] to perform the following steps:
                1. If *pendingTasks* is not −1:
                    1. Set *pendingTasks* to −1.

2. Reject *promise* with *script*'s error to rethrow[p1153].

2. Abort these steps.

3. If *addedSuccessfully* is false:

    1. Append *moduleURLRecord* to *workletInstance*'s added modules list[p1345].

    2. Set *addedSuccessfully* to true.

4. Run a module script[p1165] given *script*.

5. Queue a global task[p1195] on the networking task source[p1204] given *workletInstance*'s relevant global object[p1151] to perform the following steps:

    1. If *pendingTasks* is not −1:

        1. Set *pendingTasks* to *pendingTasks* − 1.

        2. If *pendingTasks* is 0, then resolve *promise*.

7. Return *promise*.

11.3.3 The worklet's lifetime §[p1347]

The lifetime of a `Worklet`[p1345] has no special considerations; it is tied to the object it belongs to, such as the `Window`[p965].

Each `Document`[p135] has a **worklet global scopes**, which is a set of `WorkletGlobalScope`[p1342]s, initially empty.

The lifetime of a `WorkletGlobalScope`[p1342] is, at a minimum, tied to the `Document`[p135] whose worklet global scopes[p1347] contain it. In particular, destroying[p1116] the `Document`[p135] will terminate[p1344] the corresponding `WorkletGlobalScope`[p1342] and allow it to be garbage-collected.

Additionally, user agents may, at any time, terminate[p1344] a given `WorkletGlobalScope`[p1342], unless the specification defining the corresponding worklet type says otherwise. For example, they might terminate them if the worklet agent[p1140]'s event loop[p1193] has no tasks[p1194] 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[p1343] `WorkletGlobalScope`[p1342]s for a given worklet type. For example, they might create them during specific processes that call upon worklet code, as in the example[p1342].

12 Web storage §p13 48

12.1 Introduction §p13 48

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. [COOKIES]p1580

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 noticing.

To address this, this specification introduces the `sessionStorage`p1351 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:

```
<label>
 <input type="checkbox" onchange="sessionStorage.insurance = checked ? 'true' : ''">
  I want insurance on this trip.
</label>
```

A later page could then check, from script, whether the user had checked the checkbox or not:

```
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`p1352 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:

```
<p>
  You have viewed this page
  <span id="count">an untold number of</span>
  time(s).
</p>
<script>
  if (!localStorage.pageLoadCount)
    localStorage.pageLoadCount = 0;
```

```
  localStorage.pageLoadCount = parseInt(localStorage.pageLoadCount) + 1;
  document.getElementById('count').textContent = localStorage.pageLoadCount;
</script>
```

Each site has its own separate storage area.

⚠Warning!

The localStorage[p1352] getter provides access to shared state. This specification does not define the interaction with other agent clusters 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 §[p13 49]

12.2.1 The Storage[p1349] interface §[p13 49]

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[p1350]**

Returns the number of key/value pairs.

***storage*.key[p1350] (*n*)**

Returns the name of the *n*th key, or null if *n* is greater than or equal to the number of key/value pairs.

***value* = *storage*.getItem[p1350] (*key*)**

Returns the current value associated with the given *key*, or null if the given *key* does not exist.

***value* = *storage*[*key*]**

Returns the current value associated with the given *key*, or undefined if the given *key* does not exist.

***storage*.setItem[p1350] (*key*, *value*)**

storage*[*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 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[p1576] event on Window[p965] objects holding an equivalent Storage[p1349] object.

***storage*.removeItem[p1351] (*key*)**

delete *storage*[*key*]

Removes the key/value pair with the given *key*, if a key/value pair with the given *key* exists.

Dispatches a storage[p1576] event on Window[p965] objects holding an equivalent Storage[p1349] object.

***storage*.clear[p1351]()**

Removes all key/value pairs, if there are any.

Dispatches a `storage`[p1576] event on `Window`[p965] objects holding an equivalent `Storage`[p1349] object.

A `Storage`[p1349] object has an associated:

map

A storage proxy map.

type

"`local`" or "`session`".

To **reorder** a `Storage`[p1349] object *storage*, reorder *storage*'s map[p1350]'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`[p1349] object *storage*, given a *key*, *oldValue*, and *newValue*, run these steps:

1. Let *thisDocument* be *storage*'s relevant global object[p1151]'s associated `Document`[p966].
2. Let *url* be the serialization of *thisDocument*'s URL.
3. Let *remoteStorages* be all `Storage`[p1349] objects excluding *storage* whose:
   - type[p1350] is *storage*'s type[p1350]
   - relevant settings object[p1151]'s origin[p939] is same origin[p940] with *storage*'s relevant settings object[p1151]'s origin[p939]

   and, if type[p1350] is "`session`", whose relevant settings object[p1151]'s associated `Document`[p966]'s node navigable[p1036]'s traversable navigable[p1037] is *thisDocument*'s node navigable[p1036]'s traversable navigable[p1037].
4. For each *remoteStorage* of *remoteStorages*: queue a global task[p1195] on the DOM manipulation task source[p1203] given *remoteStorage*'s relevant global object[p1151] to fire an event named `storage`[p1576] at *remoteStorage*'s relevant global object[p1151], using `StorageEvent`[p1352], with `key`[p1353] initialized to *key*, `oldValue`[p1353] initialized to *oldValue*, `newValue`[p1353] initialized to *newValue*, `url`[p1353] initialized to *url*, and `storageArea`[p1353] initialized to *remoteStorage*.

   Note

   The `Document`[p135] object associated with the resulting task[p1194] is not necessarily fully active[p1051], but events fired on such objects are ignored by the event loop[p1193] until the `Document`[p135] becomes fully active[p1051] again.

The **`length`** getter steps are to return this's map[p1350]'s size.

The **`key(index)`** method steps are:

1. If *index* is greater than or equal to this's map[p1350]'s size, then return null.
2. Let *keys* be the result of running get the keys on this's map[p1350].
3. Return *keys*[*index*].

The supported property names on a `Storage`[p1349] object *storage* are the result of running get the keys on *storage*'s map[p1350].

The **`getItem(key)`** method steps are:

1. If this's map[p1350][*key*] does not exist, then return null.
2. Return this's map[p1350][*key*].

The **`setItem(key, value)`** method steps are:

1. Let *oldValue* be null.

2. Let *reorder* be true.
3. If this's map[p1350][*key*] exists:
    1. Set *oldValue* to this's map[p1350][*key*].
    2. If *oldValue* is *value*, then return.
    3. Set *reorder* to false.
4. If *value* cannot be stored, then throw a `QuotaExceededError`.
5. Set this's map[p1350][*key*] to *value*.
6. If *reorder* is true, then reorder[p1350] this.
7. Broadcast[p1350] this with *key*, *oldValue*, and *value*.

The **`removeItem(key)`** method steps are:

1. If this's map[p1350][*key*] does not exist, then return.
2. Set *oldValue* to this's map[p1350][*key*].
3. Remove this's map[p1350][*key*].
4. Reorder[p1350] this.
5. Broadcast[p1350] this with *key*, *oldValue*, and null.

The **`clear()`** method steps are:

1. If this's map[p1350] is empty, then return.
2. Clear this's map[p1350].
3. Broadcast[p1350] this with null, null, and null.

12.2.2 The `sessionStorage`[p1351] getter §[p13 51]

IDL

```
interface mixin WindowSessionStorage {
  readonly attribute Storage sessionStorage;
};
Window includes WindowSessionStorage;
```

For web developers (non-normative)

***window*.sessionStorage**[p1351]

Returns the `Storage`[p1349] object associated with that *window*'s origin's session storage area.

Throws a `"SecurityError"` `DOMException` if the `Document`[p135]'s origin is an opaque origin[p939] 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`[p135] object has an associated **session storage holder**, which is null or a `Storage`[p1349] object. It is initially null.

The **`sessionStorage`** getter steps are:

1. If this's associated `Document`[p966]'s session storage holder[p1351] is non-null, then return this's associated `Document`[p966]'s session storage holder[p1351].
2. Let *map* be the result of running obtain a session storage bottle map with this's relevant settings object[p1151] and "`sessionStorage`".
3. If *map* is failure, then throw a `"SecurityError"` `DOMException`.
4. Let *storage* be a new `Storage`[p1349] object whose map[p1350] is *map*.

5. Set this's associated Document[p966]'s session storage holder[p1351] to *storage*.

6. Return *storage*.

Note

After creating a new auxiliary browsing context and document[p1048], the session storage is copied[p1038] over.

12.2.3 The localStorage[p1352] getter §[p13 52]

IDL

```
interface mixin WindowLocalStorage {
  readonly attribute Storage localStorage;
};
Window includes WindowLocalStorage;
```

For web developers (non-normative)

***window*.localStorage[p1352]**

Returns the Storage[p1349] object associated with *window*'s origin's local storage area.

Throws a "SecurityError" DOMException if the Document[p135]'s origin is an opaque origin[p939] 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[p135] object has an associated **local storage holder**, which is null or a Storage[p1349] object. It is initially null.

The **localStorage** getter steps are:

1. If this's associated Document[p966]'s local storage holder[p1352] is non-null, then return this's associated Document[p966]'s local storage holder[p1352].

2. Let *map* be the result of running obtain a local storage bottle map with this's relevant settings object[p1151] and "localStorage".

3. If *map* is failure, then throw a "SecurityError" DOMException.

4. Let *storage* be a new Storage[p1349] object whose map[p1350] is *map*.

5. Set this's associated Document[p966]'s local storage holder[p1352] to *storage*.

6. Return *storage*.

12.2.4 The StorageEvent[p1352] interface §[p13 52]

IDL

```
[Exposed=Window]
interface StorageEvent : Event {
  constructor(DOMString type, optional StorageEventInit eventInitDict = {});

  readonly attribute DOMString? key;
  readonly attribute DOMString? oldValue;
  readonly attribute DOMString? newValue;
  readonly attribute USVString url;
  readonly attribute Storage? storageArea;

  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);
};

dictionary StorageEventInit : EventInit {
  DOMString? key = null;
```

```
  DOMString? oldValue = null;
  DOMString? newValue = null;
  USVString url = "";
  Storage? storageArea = null;
};
```

For web developers (non-normative)

***event*.key**[p1353]

Returns the key of the storage item being changed.

***event*.oldValue**[p1353]

Returns the old value of the key of the storage item whose value is being changed.

***event*.newValue**[p1353]

Returns the new value of the key of the storage item whose value is being changed.

***event*.url**[p1353]

Returns the URL of the document whose storage item changed.

***event*.storageArea**[p1353]

Returns the `Storage`[p1349] object that was affected.

The **key**, **oldValue**, **newValue**, **url**, and **storageArea** attributes must return the values they were initialized to.

The **initStorageEvent(*type*, *bubbles*, *cancelable*, *key*, *oldValue*, *newValue*, *url*, *storageArea*)** method must initialize the event in a manner analogous to the similarly-named `initEvent()` method. [DOM][p1582]

12.3 Privacy §[p13 53]

12.3.1 User tracking §[p13 53]

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`[p1352] objects to scripts originating at the domain of the active document[p1036] of the top-level traversable[p1037], for instance denying access to the API for pages from other domains running in `iframe`[p404]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 navigables[p1035] 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][p1580]

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.

Origin-tracking of stored data

User agents may record the origins[p939] 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 §[p13 54]

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 §[p13 54]

12.4.1 DNS spoofing attacks §[p13 54]

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 §[p13 54]

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 §[p13 55]

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[p939] model described in this specification is important for user security.

13 The HTML syntax §p13 56

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 p1484".

13.1 Writing HTML documents §p13 56

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 p1367 and ASCII whitespace.
3. A DOCTYPE p1356.
4. Any number of comments p1367 and ASCII whitespace.
5. The document element, in the form of an `html` p179 element p1357.
6. Any number of comments p1367 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 p206 are to be serialized, as discussed in the section on that topic.

Note

ASCII whitespace before the `html` p179 element, at the start of the `html` p179 element and before the `head` p180 element, will be dropped when the document is parsed; ASCII whitespace after the `html` p179 element will be parsed as if it were at the end of the `body` p212 element. Thus, ASCII whitespace around the document element does not roundtrip.

It is suggested that newlines be inserted after the DOCTYPE, after any comments that are before the document element, after the `html` p179 element's start tag (if it is not omitted p1360), and after any comments that are inside the `html` p179 element but before the `head` p180 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 §p13 56

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[p1357].
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`[p100]".
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[p1357] should not be used unless the document is generated from a system that cannot output the shorter string.

13.1.2 Elements §[p13] 57

There are six different kinds of **elements**: void elements[p1357], the `template` element[p1357], raw text elements[p1357], escapable raw text elements[p1357], foreign elements[p1357], and normal elements[p1357].

Void elements

`area`[p489], `base`[p182], `br`[p310], `col`[p506], `embed`[p413], `hr`[p241], `img`[p359], `input`[p539], `link`[p184], `meta`[p196], `source`[p355], `track`[p427], `wbr`[p311]

The `template` element

`template`[p705]

Raw text elements

`script`[p685], `style`[p207]

Escapable raw text elements

`textarea`[p606], `title`[p181]

Foreign elements

Elements from the MathML namespace and the SVG namespace.

Normal elements

All other allowed HTML elements[p46] are normal elements.

Tags are used to delimit the start and end of elements in the markup. Raw text[p1357], escapable raw text[p1357], and normal[p1357] elements have a start tag[p1358] to indicate where they begin, and an end tag[p1359] to indicate where they end. The start and end tags of certain normal elements[p1357] can be omitted[p1360], as described below in the section on optional tags[p1360]. Those that cannot be omitted must not be omitted. Void elements[p1357] only have a start tag; end tags must not be specified for void elements[p1357]. Foreign elements[p1357] 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[p155] of the element must be placed between just after the start tag (which might be implied, in certain cases[p1360]) and just before the end tag (which again, might be implied in certain cases[p1360]). The exact allowed contents of each individual element depend on the content model[p155] 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[p1357] 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[p1357] can have template contents[p707], but such template contents[p707] are not children of the `template`[p705] element itself. Instead, they are stored in a `DocumentFragment` associated with a different `Document`[p135] — without a browsing context[p1046] — so as to avoid the `template`[p705] contents interfering with the main `Document`[p135]. The markup for the template contents[p707] of a `template`[p705] element is placed just after the `template`[p705] element's start tag and just before `template`[p705] element's end tag (as with other elements), and may consist of any text[p1366], character references[p1366], elements[p1357], and comments[p1367], but the text must not contain the character U+003C LESS-THAN SIGN (<) or an ambiguous ampersand[p1367].

Raw text elements[p1357] can have text[p1366], though it has restrictions[p1366] described below.

Escapable raw text elements[p1357] can have text[p1366] and character references[p1366], but the text must not contain an ambiguous ampersand[p1367]. There are also further restrictions[p1366] described below.

Foreign elements[p1357] 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[p1357] whose start tag is *not* marked as self-closing can have text[p1366], character references[p1366], CDATA sections[p1367], other elements[p1357], and comments[p1367], but the text must not contain the character U+003C LESS-THAN SIGN (<) or an ambiguous ampersand[p1367].

Note

The HTML syntax does not support namespace declarations, even in foreign elements[p1357].

For instance, consider the following HTML fragment:

```
<p>
 <svg>
  <metadata>
   <!-- this is invalid -->
   <cdr:license xmlns:cdr="https://www.example.com/cdr/metadata" name="MIT"/>
  </metadata>
 </svg>
</p>
```

The innermost element, `cdr:license`, is actually in the SVG namespace, as the "`xmlns:cdr`" attribute has no effect (unlike in XML). In fact, as the comment in the fragment above says, the fragment is actually non-conforming. This is because SVG 2 does not define any elements called "`cdr:license`" in the SVG namespace.

Normal elements[p1357] can have text[p1366], character references[p1366], other elements[p1357], and comments[p1367], but the text must not contain the character U+003C LESS-THAN SIGN (<) or an ambiguous ampersand[p1367]. Some normal elements[p1357] also have yet more restrictions[p1366] 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[p1357], 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 §[p1358]

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[p1358].
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[p1359] 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[p1358] 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[p1359] below.)
6. Then, if the element is one of the void elements[p1357], or if the element is a foreign element[p1357], then there may be a single U+002F SOLIDUS character (/), which on foreign elements[p1357] marks the start tag as self-closing. On void elements[p1357], it

does not mark the start tag as self-closing but instead is unnecessary and has no effect of any kind. For such void elements, it should be used only with caution — especially since, if directly preceded by an unquoted attribute value[p1359], it becomes part of the attribute value rather than being discarded by the parser.

7. Finally, start tags must be closed by a U+003E GREATER-THAN SIGN character (>).

13.1.2.2 End tags §[p1359]

End tags must have the following format:

1. The first character of an end tag must be a U+003C LESS-THAN SIGN character (<).
2. The second character of an end tag must be a U+002F SOLIDUS character (/).
3. The next few characters of an end tag must be the element's tag name[p1358].
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-THAN SIGN character (>).

13.1.2.3 Attributes §[p1359]

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 ("), U+0027 ('), U+003E (>), U+002F (/), U+003D (=), and noncharacters. In the HTML syntax, attribute names, even those for foreign elements[p1357], may be written with any mix of ASCII lower and ASCII upper alphas.

Attribute values are a mixture of text[p1366] and character references[p1366], except with the additional restriction that the text cannot contain an ambiguous ampersand[p1367].

Attributes can be specified in four different ways:

Empty attribute syntax

Just the attribute name[p1359]. The value is implicitly the empty string.

Example

In the following example, the `disabled`[p630] attribute is given with the empty attribute syntax:

```
<input disabled>
```

If an attribute using the empty attribute syntax is to be followed by another attribute, then there must be ASCII whitespace separating the two.

Unquoted attribute value syntax

The attribute name[p1359], 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[p1359], 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-THAN SIGN characters (<), U+003E GREATER-THAN SIGN characters (>), or U+0060 GRAVE ACCENT characters (`), and must not be the empty string.

Example

In the following example, the `value`[p543] attribute is given with the unquoted attribute value syntax:

```
<input value=yes>
```

If an attribute using the unquoted attribute syntax is to be followed by another attribute or by the optional U+002F SOLIDUS character (/) allowed in step 6 of the start tag[p1358] syntax above, then there must be ASCII whitespace separating the two.

Single-quoted attribute value syntax

The attribute name[p1359], 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[p1359], 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[p541] attribute is given with the single-quoted attribute value syntax:

```
<input type='checkbox'>
```

If an attribute using the single-quoted attribute syntax is to be followed by another attribute, then there must be ASCII whitespace separating the two.

Double-quoted attribute value syntax

The attribute name[p1359], 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[p1359], 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[p628] attribute is given with the double-quoted attribute value syntax:

```
<input name="be evil">
```

If an attribute using the double-quoted attribute syntax is to be followed by another attribute, then there must be 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[p1357] has one of the namespaced attributes given by the local name and namespace of the first and second cells of a row from the following table, it must be written using the name given by the third cell from the same row.

Local name	Namespace	Attribute name
actuate	XLink namespace	xlink:actuate
arcrole	XLink namespace	xlink:arcrole
href	XLink namespace	xlink:href
role	XLink namespace	xlink:role
show	XLink namespace	xlink:show
title	XLink namespace	xlink:title
type	XLink namespace	xlink:type
lang	XML namespace	xml:lang
space	XML namespace	xml:space
xmlns	XMLNS namespace	xmlns
xlink	XMLNS namespace	xmlns:xlink

No other namespaced attribute can be expressed in the HTML syntax[p1356].

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 §[p13 60]

Certain tags can be **omitted**.

Note

Omitting an element's start tag[p1358] 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[p179] element, even if the string <html> doesn't appear anywhere in the markup.

An html[p179] element's start tag[p1358] may be omitted if the first thing inside the html[p179] element is not a comment[p1367].

Example

For example, in the following case it's ok to remove the "<html>" tag:

```
<!DOCTYPE HTML>
<html>
  <head>
    <title>Hello</title>
  </head>
  <body>
    <p>Welcome to this example.</p>
  </body>
</html>
```

Doing so would make the document look like this:

```
<!DOCTYPE HTML>

  <head>
    <title>Hello</title>
  </head>
  <body>
    <p>Welcome to this example.</p>
  </body>
</html>
```

This has the exact same DOM. In particular, note that whitespace around the document element is ignored by the parser. The following example would also have the exact same DOM:

```
<!DOCTYPE HTML><head>
    <title>Hello</title>
  </head>
  <body>
    <p>Welcome to this example.</p>
  </body>
</html>
```

However, in the following example, removing the start tag moves the comment to before the html[p179] element:

```
<!DOCTYPE HTML>
<html>
  <!-- where is this comment in the DOM? -->
  <head>
    <title>Hello</title>
  </head>
  <body>
    <p>Welcome to this example.</p>
  </body>
</html>
```

With the tag removed, the document actually turns into the same as this:

```
<!DOCTYPE HTML>
```

```
<!-- where is this comment in the DOM? -->
<html>
  <head>
    <title>Hello</title>
  </head>
  <body>
    <p>Welcome to this example.</p>
  </body>
</html>
```

This is why the tag can only be removed if it is not followed by a comment: removing the tag when there is a comment there changes the document's resulting parse tree. Of course, if the position of the comment does not matter, then the tag can be omitted, as if the comment had been moved to before the start tag in the first place.

An `html`[p179] element's end tag[p1359] may be omitted if the `html`[p179] element is not immediately followed by a comment[p1367].

A `head`[p180] element's start tag[p1358] may be omitted if the element is empty, or if the first thing inside the `head`[p180] element is an element.

A `head`[p180] element's end tag[p1359] may be omitted if the `head`[p180] element is not immediately followed by ASCII whitespace or a comment[p1367].

A `body`[p212] element's start tag[p1358] may be omitted if the element is empty, or if the first thing inside the `body`[p212] element is not ASCII whitespace or a comment[p1367], except if the first thing inside the `body`[p212] element is a `meta`[p196], `noscript`[p703], `link`[p184], `script`[p685], `style`[p207], or `template`[p705] element.

A `body`[p212] element's end tag[p1359] may be omitted if the `body`[p212] element is not immediately followed by a comment[p1367].

Example

Note that in the example above, the `head`[p180] element start and end tags, and the `body`[p212] 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`[p212] and `html`[p179] element end tags could be omitted without trouble; any spaces after those get parsed into the `body`[p212] 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>
```

```
<p>Welcome to this example.</p>
```

This would be equivalent to this document, with the omitted tags shown in their parser-implied positions; the only whitespace text node that results from this is the newline at the end of the head element:

```
<!DOCTYPE HTML>
<html><head><title>Hello</title>
</head><body><p>Welcome to this example.</p></body></html>
```

An li element's end tag may be omitted if the li element is immediately followed by another li element or if there is no more content in the parent element.

A dt element's end tag may be omitted if the dt element is immediately followed by another dt element or a dd element.

A dd element's end tag may be omitted if the dd element is immediately followed by another dd element or a dt element, or if there is no more content in the parent element.

A p element's end tag may be omitted if the p element is immediately followed by an address, article, aside, blockquote, details, dialog, div, dl, fieldset, figcaption, figure, footer, form, h1, h2, h3, h4, h5, h6, header, hgroup, hr, main, menu, nav, ol, p, pre, search, 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.

Example

We can thus simplify the earlier example further:

```
<!DOCTYPE HTML><title>Hello</title><p>Welcome to this example.
```

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, if it is immediately followed by an hr 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, if it is immediately followed by an optgroup element, if it is immediately followed by an hr 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[p509] element's end tag[p1359] may be omitted if there is no more content in the parent element.

A tr[p510] element's end tag[p1359] may be omitted if the tr[p510] element is immediately followed by another tr[p510] element, or if there is no more content in the parent element.

A td[p512] element's end tag[p1359] may be omitted if the td[p512] element is immediately followed by a td[p512] or th[p513] element, or if there is no more content in the parent element.

A th[p513] element's end tag[p1359] may be omitted if the th[p513] element is immediately followed by a td[p512] or th[p513] element, or if there is no more content in the parent element.

Example

The ability to omit all these table-related tags makes table markup much terser.

Take this example:

```
<table>
 <caption>37547 TEE Electric Powered Rail Car Train Functions (Abbreviated)</caption>
 <colgroup><col><col><col></colgroup>
 <thead>
  <tr>
   <th>Function</th>
   <th>Control Unit</th>
   <th>Central Station</th>
  </tr>
 </thead>
 <tbody>
  <tr>
   <td>Headlights</td>
   <td>✔</td>
   <td>✔</td>
  </tr>
  <tr>
   <td>Interior Lights</td>
   <td>✔</td>
   <td>✔</td>
  </tr>
  <tr>
   <td>Electric locomotive operating sounds</td>
   <td>✔</td>
   <td>✔</td>
  </tr>
  <tr>
   <td>Engineer's cab lighting</td>
   <td></td>
   <td>✔</td>
  </tr>
  <tr>
   <td>Station Announcements - Swiss</td>
   <td></td>
   <td>✔</td>
  </tr>
 </tbody>
</table>
```

The exact same table, modulo some whitespace differences, could be marked up as follows:

```
<table>
 <caption>37547 TEE Electric Powered Rail Car Train Functions (Abbreviated)
```

```
 <colgroup><col><col><col>
 <thead>
  <tr>
   <th>Function
   <th>Control Unit
   <th>Central Station
 <tbody>
  <tr>
   <td>Headlights
   <td>✔
   <td>✔
  <tr>
   <td>Interior Lights
   <td>✔
   <td>✔
  <tr>
   <td>Electric locomotive operating sounds
   <td>✔
   <td>✔
  <tr>
   <td>Engineer's cab lighting
   <td>
   <td>✔
  <tr>
   <td>Station Announcements - Swiss
   <td>
   <td>✔
</table>
```

Since the cells take up much less room this way, this can be made even terser by having each row on one line:

```
<table>
 <caption>37547 TEE Electric Powered Rail Car Train Functions (Abbreviated)
 <colgroup><col><col><col>
 <thead>
  <tr> <th>Function                              <th>Control Unit     <th>Central Station
 <tbody>
  <tr> <td>Headlights                           <td>✔                <td>✔
  <tr> <td>Interior Lights                      <td>✔                <td>✔
  <tr> <td>Electric locomotive operating sounds <td>✔                <td>✔
  <tr> <td>Engineer's cab lighting              <td>                 <td>✔
  <tr> <td>Station Announcements - Swiss        <td>                 <td>✔
</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[p1358] 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:

```
<!DOCTYPE HTML><title>Hello</title><p>Welcome to this example.
```

If the body[p212] element in this example had to have a class[p162] attribute and the html[p179] element had to have a lang[p165] attribute, the markup would have to become:

```
<!DOCTYPE HTML><html lang="en"><title>Hello</title><body class="demo"><p>Welcome to this example.
```

Note

This section assumes that the document is conforming, in particular, that there are no content model[p155] violations. Omitting tags in the fashion described in this section in a document that does not conform to the content models[p155] described in this specification is likely to result in unexpected DOM differences (this is, in part, what the content models are designed to avoid).

13.1.2.5 Restrictions on content models §[p13 66]

For historical reasons, certain elements have extra restrictions beyond even the restrictions given by their content model.

A `table`[p496] element must not contain `tr`[p510] elements, even though these elements are technically allowed inside `table`[p496] elements according to the content models described in this specification. (If a `tr`[p510] element is put inside a `table`[p496] in the markup, it will in fact imply a `tbody`[p507] start tag before it.)

A single newline[p1366] may be placed immediately after the start tag[p1358] of `pre`[p242] and `textarea`[p606] elements. This does not affect the processing of the element. The otherwise optional newline[p1366] *must* be included if the element's contents themselves start with a newline[p1366] (because otherwise the leading newline in the contents would be treated like the optional newline, and ignored).

Example

The following two `pre`[p242] blocks are equivalent:

```
<pre>Hello</pre>
```

```
<pre>
Hello</pre>
```

13.1.2.6 Restrictions on the contents of raw text and escapable raw text elements §[p13 66]

The text in raw text[p1357] and escapable raw text elements[p1357] 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 (/).

13.1.3 Text §[p13 66]

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.

13.1.3.1 Newlines §[p13 66]

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[p1366] 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[p1366].

13.1.4 Character references §[p13 66]

In certain cases described in other sections, text[p1366] may be mixed with **character references**. These can be used to escape characters that couldn't otherwise legally be included in text[p1366].

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[p1474] 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[p1474] section.

13.1.5 CDATA sections §[p13] 67

CDATA sections must consist of the following components, in this order:

1. The string "`<![CDATA[`".
2. Optionally, text[p1366], 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:

```
<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 §[p13] 67

Comments must have the following format:

1. The string "`<!--`".
2. Optionally, text[p1366], 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[p1366] is allowed to end with the string "<!", as in `<!--My favorite operators are > and <!-->`.

13.1.7 Processing instructions §[p1368]

A processing instruction marks a location in a document for further processing.

Processing instructions must have the following format:

1. "<?".
2. A target[p1368].
3. Either of the following two options:
    - Zero or more ASCII whitespace.
    - One or more ASCII whitespace, followed by data[p1368].
4. Optionally, a U+003F (?).
5. A U+003E (>).

The **target** consists of ASCII alphanumeric, U+002D (-), and U+005F (_) code points. It must start with an ASCII alpha or U+005F (_) and must not be an ASCII case-insensitive match for "`xml`" or "`xml-stylesheet`".

The **data** is text[p1366], with the additional restriction that the text must not contain U+003E (>), nor end with U+003F (?).

Note

Processing instructions in the HTML syntax have some notable differences to processing instructions in XML: [XML][p1588]

- *In HTML, a processing instruction can end with ">" or "?>", while in XML it can only end with "?>".*
- *In HTML, "<?xml ...?>" and "<?xml-stylesheet ...?>" specifically are not parsed as processing instructions but instead treated as bogus comments[p1401]. This is for compatibility with existing HTML content, where such syntax is relatively common.*
- *HTML does not support the full Name production for processing instruction targets[p1368]. A much more limited subset is allowed, as noted above. [XML][p1588]*

13.2 Parsing HTML documents §[p1368]

This section only applies to user agents, data mining tools, and conformance checkers.

Note

The rules for parsing XML documents into DOM trees are covered by the next section, entitled "The XML syntax[p1484]".

User agents must use the parsing rules described in this section to generate the DOM trees from `text/html`[p1546] 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.

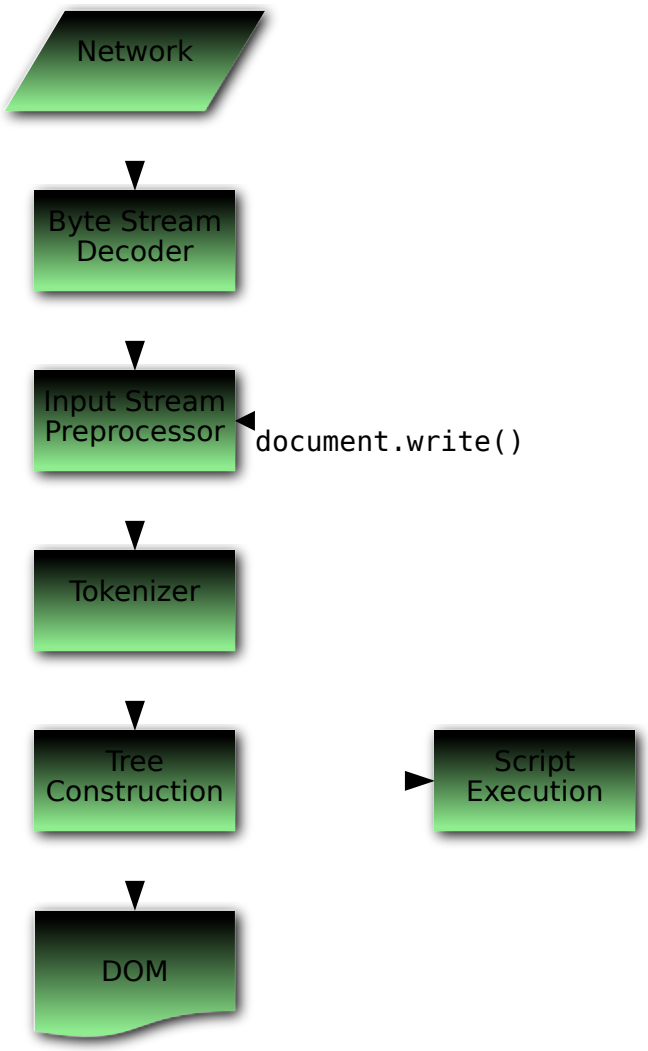
Some earlier versions of HTML (in particular from HTML2 to HTML4) were based on SGML and used SGML parsing rules. However, few (if any) web browsers ever implemented true SGML parsing for HTML documents; the only user agents to strictly handle HTML as an SGML application have historically been validators. 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.

For the purposes of conformance checkers, if a resource is determined to be in the HTML syntax[p1356], then it is an HTML document.

Note

As stated in the terminology section[p46], references to element types[p46] that do not explicitly specify a namespace always refer to elements in the HTML namespace. For example, if the spec talks about "a `menu`[p250] element", then that is an element with the local name "`menu`", the namespace "`http://www.w3.org/1999/xhtml`", and the interface `HTMLMenuElement`[p250]. Where possible, references to such elements are hyperlinked to their definition.

13.2.1 Overview of the parsing model §[p13] 69



The input to the HTML parsing process consists of a stream of code points, which is passed through a tokenization[p1387] stage followed by a tree construction[p1417] stage. The output is a `Document`[p135] object.

Note

Implementations that do not support scripting[p49] do not have to actually create a DOM `Document`[p135] object, but the DOM tree in such cases is still used as the model for the rest of the specification.

In the common case, the data handled by the tokenization stage comes from the network, but it can also come from script[p1220] running in the user agent, e.g. using the `document.write()`[p1223] 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:

```
...
<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 §p13 70

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[p1370]. 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[p1459] at the first parse error[p1370] 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.

Code	Description
abrupt-closing-of-empty-comment	This error occurs if the parser encounters an empty comment[p1367] that is abruptly closed by a U+003E (>) code point (i.e., `<!-->` or `<!--->`). The parser behaves as if the comment is closed correctly.
abrupt-doctype-public-identifier	This error occurs if the parser encounters a U+003E (>) code point in the DOCTYPE[p1356] public identifier (e.g., `<!DOCTYPE html PUBLIC "foo>`). In such a case, if the DOCTYPE is correctly placed as a document preamble, the parser sets the Document[p135] to quirks mode.
abrupt-doctype-system-identifier	This error occurs if the parser encounters a U+003E (>) code point in the DOCTYPE[p1356] system identifier (e.g., `<!DOCTYPE html PUBLIC "-//W3C//DTD HTML 4.01//EN" "foo>`). In such a case, if the DOCTYPE is correctly placed as a document preamble, the parser sets the Document[p135] to quirks mode.
absence-of-digits-in-numeric-character-reference	This error occurs if the parser encounters a numeric character reference[p1366] that doesn't contain any digits (e.g., `&#qux;`). In this case the parser doesn't resolve the character reference.
cdata-in-html-content	This error occurs if the parser encounters a CDATA section[p1367] outside of foreign content (SVG or MathML). The parser treats such CDATA sections (including leading "`[CDATA[`" and trailing "`]]`") as comments.
character-reference-outside-unicode-range	This error occurs if the parser encounters a numeric character reference[p1366] 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.
control-character-in-input-stream	This error occurs if the input stream[p1382] 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.
control-character-reference	This error occurs if the parser encounters a numeric character reference[p1366] 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[p1416].

Code	Description
disallowed-processing-instruction-target	This error occurs if the parser encounters a processing instruction target[p1368] that is an ASCII case-insensitive match for "`xml`" or "`xml-stylesheet`". The preceding U+003F (?) and all content that follows up to a U+003E (>) (if present) or to the end of the input stream[p1382] is treated as a comment.
duplicate-attribute	This error occurs if the parser encounters an attribute[p1359] in a tag that already has an attribute with the same name. The parser ignores all such duplicate occurrences of the attribute.
end-tag-with-attributes	This error occurs if the parser encounters an end tag[p1359] with attributes[p1359]. Attributes in end tags are ignored and do not make their way into the DOM.
end-tag-with-trailing-solidus	This error occurs if the parser encounters an end tag[p1359] that has a U+002F (/) code point right before the closing U+003E (>) code point (e.g., `</div/>`). Such a tag is treated as a regular end tag.
eof-before-tag-name	This error occurs if the parser encounters the end of the input stream[p1382] where a tag name is expected. In this case the parser treats the beginning of a start tag[p1358] (i.e., `<`) or an end tag[p1359] (i.e., `</`) as text content.
eof-in-cdata	This error occurs if the parser encounters the end of the input stream[p1382] in a CDATA section[p1367]. The parser treats such CDATA sections as if they are closed immediately before the end of the input stream.
eof-in-comment	This error occurs if the parser encounters the end of the input stream[p1382] in a comment[p1367]. The parser treats such comments as if they are closed immediately before the end of the input stream.
eof-in-doctype	This error occurs if the parser encounters the end of the input stream in a DOCTYPE[p1356]. In such a case, if the DOCTYPE is correctly placed as a document preamble, the parser sets the `Document`[p135] to quirks mode.
eof-in-processing-instruction	This error occurs if the parser encounters the end of the input stream[p1382] in a processing instruction[p1368] (e.g., `<?` or `<?marker name=`). Such processing instructions are ignored.
eof-in-script-html-comment-like-text	This error occurs if the parser encounters the end of the input stream[p1382] in text that resembles an HTML comment[p1367] inside `script`[p685] element content (e.g., `<script><!-- foo`). Note *Syntactic structures that resemble HTML comments in `script`[p685] 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[p700]. [JAVASCRIPT][p1583]*
eof-in-tag	This error occurs if the parser encounters the end of the input stream[p1382] in a start tag[p1358] or an end tag[p1359] (e.g., `<div id=`). Such a tag is ignored.
incorrectly-closed-comment	This error occurs if the parser encounters a comment[p1367] that is closed by the "`--!>`" code point sequence. The parser treats such comments as if they are correctly closed by the "`-->`" code point sequence.
incorrectly-opened-comment	This error occurs if the parser encounters the "`<!`" code point sequence that is not immediately followed by two U+002D (-) code points and that is not the start of a DOCTYPE[p1356] or a CDATA section[p1367]. All content that follows the "`<!`" code point sequence up to a U+003E (>) code point (if present) or to the end of the input stream[p1382] is treated as a comment. Note *One possible cause of this error is using an XML markup declaration (e.g., `<!ELEMENT br EMPTY>`) in HTML.*
invalid-character-sequence-after-doctype-name	This error occurs if the parser encounters any code point sequence other than "`PUBLIC`" and "`SYSTEM`" keywords after a DOCTYPE[p1356] 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, and if the parser cannot change the mode flag[p1424] is false, sets the `Document`[p135] to quirks mode.
invalid-first-character-of-processing-instruction-target	This error occurs if the parser encounters a code point that is not an ASCII alpha or U+005F (_) where the first code point of a processing instruction target[p1368] is expected. The preceding U+003F (?) and all content that follows up to a U+003E (>) (if present) or to the end of the input stream[p1382] is treated as a comment.
invalid-first-character-of-tag-name	This error occurs if the parser encounters a code point that is not an ASCII alpha where the first code point of a start tag[p1358] name or an end tag[p1359] name is expected. If a start tag was expected such code point and a preceding U+003C (<) 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 (>) code point (if present) or to the end of the input stream[p1382] is treated as a comment. Example For example, consider the following markup:

Code	Description

	<42></42>
	
	This will be parsed into:
	
	└html[p179]
	  ├head[p180]
	  └body[p212]
	  ├#text: *42*
	  └#comment: *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.

invalid-processing-instruction-target	This error occurs if the parser encounters a code point that is not an ASCII alphanumeric, U+002D (-), or U+005F (_) where a processing instruction target[p1368] is expected. The preceding U+003F (?) and all content that follows up to a U+003E (>) (if present) or to the end of the input stream[p1382] is treated as a comment.
missing-attribute-value	This error occurs if the parser encounters a U+003E (>) code point where an attribute[p1359] value is expected (e.g., `<div id=>`). The parser treats the attribute as having an empty value.
missing-doctype-name	This error occurs if the parser encounters a DOCTYPE[p1356] that is missing a name (e.g., `<!DOCTYPE>`). In such a case, if the DOCTYPE is correctly placed as a document preamble, the parser sets the `Document`[p135] to quirks mode.
missing-doctype-public-identifier	This error occurs if the parser encounters a U+003E (>) code point where start of the DOCTYPE[p1356] public identifier is expected (e.g., `<!DOCTYPE html PUBLIC >`). In such a case, if the DOCTYPE is correctly placed as a document preamble, the parser sets the `Document`[p135] to quirks mode.
missing-doctype-system-identifier	This error occurs if the parser encounters a U+003E (>) code point where start of the DOCTYPE[p1356] system identifier is expected (e.g., `<!DOCTYPE html SYSTEM >`). In such a case, if the DOCTYPE is correctly placed as a document preamble, the parser sets the `Document`[p135] to quirks mode.
missing-end-tag-name	This error occurs if the parser encounters a U+003E (>) code point where an end tag[p1359] name is expected, i.e., `</>`. The parser ignores the whole "</>" code point sequence.
missing-quote-before-doctype-public-identifier	This error occurs if the parser encounters the DOCTYPE[p1356] public identifier that is not preceded by a quote (e.g., `<!DOCTYPE html PUBLIC -//W3C//DTD HTML 4.01//EN">`). In such a case, the parser ignores the public identifier, and if the DOCTYPE is correctly placed as a document preamble, sets the `Document`[p135] to quirks mode.
missing-quote-before-doctype-system-identifier	This error occurs if the parser encounters the DOCTYPE[p1356] system identifier that is not preceded by a quote (e.g., `<!DOCTYPE html SYSTEM http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">`). In such a case, the parser ignores the system identifier, and if the DOCTYPE is correctly placed as a document preamble, sets the `Document`[p135] to quirks mode.
missing-semicolon-after-character-reference	This error occurs if the parser encounters a character reference[p1366] that is not terminated by a U+003B (;) code point. The parser behaves the same as if the character reference is terminated by the U+003B (;) code point. **Note** *Most named character references[p1474] require a terminating U+003B (;) code point. Those that don't might get resolved as a longer named character reference[p1366] in certain ambiguous scenarios.* **Example** For example, `&notin` will be parsed as "¬in", i.e., the same as if the input were `¬in`, whereas `∉` will be parsed as "∉".
missing-whitespace-after-doctype-public-keyword	This error occurs if the parser encounters a DOCTYPE[p1356] whose "PUBLIC" keyword and public identifier are not separated by ASCII whitespace. In this case the parser behaves as if ASCII whitespace is present.
missing-whitespace-after-doctype-system-keyword	This error occurs if the parser encounters a DOCTYPE[p1356] whose "SYSTEM" keyword and system identifier are not separated by ASCII whitespace. In this case the parser behaves as if ASCII whitespace is present.
missing-whitespace-before-doctype-	This error occurs if the parser encounters a DOCTYPE[p1356] whose "DOCTYPE" keyword and name are not separated by ASCII whitespace. In this case the parser behaves as if ASCII whitespace is present.

Code	Description
name	
missing-whitespace-between-attributes	This error occurs if the parser encounters attributes[p1359] that are not separated by ASCII whitespace (e.g., `<div id="foo"class="bar">`). In this case the parser behaves as if ASCII whitespace is present.
missing-whitespace-between-doctype-public-and-system-identifiers	This error occurs if the parser encounters a DOCTYPE[p1356] whose public and system identifiers are not separated by ASCII whitespace. In this case the parser behaves as if ASCII whitespace is present.
nested-comment	This error occurs if the parser encounters a nested comment[p1367] (e.g., `<!-- <!-- nested --> -->`). Such a comment will be closed by the first occurring "`-->`" code point sequence and everything that follows will be treated as markup.
noncharacter-character-reference	This error occurs if the parser encounters a numeric character reference[p1366] that references a noncharacter. The parser resolves such character references as-is.
noncharacter-in-input-stream	This error occurs if the input stream[p1382] 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.

non-void-html-element-start-tag-with-trailing-solidus	This error occurs if the parser encounters a start tag[p1358] for an element that is not in the list of void elements[p1357] 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 (>) code point. The parser behaves as if the U+002F (/) is not present.
	
	Example
	For example, consider the following markup:
	`<div/><span></span><span></span>`
	This will be parsed into:
	└ html[p179]
	  ├ head[p180]
	  └ body[p212]
	  └ div[p266]
	  ├ span[p309]
	  └ span[p309]
	
	Note
	The trailing U+002F (/) in a start tag name can be used only in foreign content to specify self-closing tags. (Self-closing tags don't exist in HTML.) It is also allowed for void elements, but doesn't have any effect in this case.

null-character-reference	This error occurs if the parser encounters a numeric character reference[p1366] that references a U+0000 NULL code point. The parser resolves such character references to a U+FFFD REPLACEMENT CHARACTER.
surrogate-character-reference	This error occurs if the parser encounters a numeric character reference[p1366] that references a surrogate. The parser resolves such character references to a U+FFFD REPLACEMENT CHARACTER.
surrogate-in-input-stream	This error occurs if the input stream[p1382] 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. Note *Surrogates can only find their way into the input stream via script APIs such as* `document.write()`[p1223].
unexpected-character-after-doctype-system-identifier	This error occurs if the parser encounters any code points other than ASCII whitespace or closing U+003E (>) after the DOCTYPE[p1356] system identifier. The parser ignores these code points.
unexpected-character-in-attribute-name	This error occurs if the parser encounters a U+0022 ("), U+0027 ('), or U+003C (<) code point in an attribute name[p1359]. The parser includes such code points in the attribute name. Note *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.* Example

Code	Description

	For example, consider the following markup:
	
	`<div foo<div>`
	
	Due to a forgotten U+003E (>) code point after `foo` the parser treats this markup as a single `div` element with a "`foo<div`" attribute.
	
	As another example of this error, consider the following markup:
	
	`<div id'bar'>`
	
	Due to a forgotten U+003D (=) 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.

unexpected-character-in-unquoted-attribute-value	This error occurs if the parser encounters a U+0022 ("), U+0027 ('), U+003C (<), U+003D (=), or U+0060 (`) code point in an unquoted attribute value. The parser includes such code points in the attribute value.  Note *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 value.*  Note *U+0060 (`) is in the list of code points that trigger this error because certain legacy user agents treat it as a quote.* Example For example, consider the following markup: `<div foo=b'ar'>` Due to a misplaced U+0027 (') code point the parser sets the value of the "`foo`" attribute to "`b'ar'`".
unexpected-equals-sign-before-attribute-name	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. Note *The common reason for this error is a forgotten attribute name.* Example For example, consider the following markup: `<div foo="bar" ="baz">` Due to a forgotten attribute name the parser treats this markup as a `div` element with two attributes: a "`foo`" attribute with a "`bar`" value and a "`="baz"`" attribute with an empty value.
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 ignored or, for security reasons, replaced with a U+FFFD REPLACEMENT CHARACTER.
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][p1588]

Usually, the encoding sniffing algorithm[p1375] defined below is used to determine the character encoding.

Given a character encoding, the bytes in the input byte stream[p1374] must be converted to characters for the tokenizer's input stream[p1382], by passing the input byte stream[p1374] and character encoding to decode.

Note

A leading Byte Order Mark (BOM) causes the character encoding argument to be ignored and will itself be skipped.

Note

Bytes or sequences of bytes in the original byte stream that did not conform to the Encoding standard (e.g. invalid UTF-8 byte sequences in a UTF-8 input byte stream) are errors that conformance checkers are expected to report. [ENCODING][p1582]

⚠Warning!

The decoder algorithms describe how to handle invalid input; for security reasons, it is imperative that those rules be followed precisely. Differences in how invalid byte sequences are handled can result in, amongst other problems, script injection vulnerabilities ("XSS").

When the HTML parser is decoding an input byte stream, it uses a character encoding and a **confidence**. The confidence is either *tentative*, *certain*, or *irrelevant*. The encoding used, and whether the confidence in that encoding is *tentative* or *certain*, is used during the parsing[p1427] to determine whether to change the encoding[p1381]. 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[p1375] is *irrelevant*.

Note

Some algorithms feed the parser by directly adding characters to the input stream[p1382] *rather than adding bytes to the input byte stream*[p1374].

13.2.3.1 Parsing with a known character encoding §[p13 75]

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[p1375] is *certain*.

13.2.3.2 Determining the character encoding §[p13 75]

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[p102] of the document) and all the bytes available so far, and returns a character encoding and a confidence[p1375] that is either *tentative* or *certain*.

1. If the result of BOM sniffing is an encoding, return that encoding with confidence[p1375] *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[p1375].

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[p1375] *certain*.

Note

Typically, user agents remember such user requests across sessions, and in some cases apply them to documents in `iframe`[p404]s 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 preparsing the source to find the encoding improves performance, as it reduces the need to throw away the data structures used when parsing upon finding the encoding information. However, if the user agent delays too long to obtain data to determine the encoding, then the cost of the delay could outweigh any performance improvements from the preparse.

Note

The authoring conformance requirements for character encoding declarations limit them to only appearing in the first 1024 bytes[p207]. 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.

4. If the transport layer specifies a character encoding, and it is supported, return that encoding with the confidence[p1375] *certain*.

5. Optionally, prescan the byte stream to determine its encoding[p1377], with the *end condition*[p1377] being when 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 returns either a character encoding or failure. If it returns a character encoding, then return the same encoding, with confidence[p1375] *tentative*.

6. If the HTML parser[p1368] for which this algorithm is being run is associated with a `Document`[p135] *d* whose container document[p1039] is non-null:

   1. Let *parentDocument* be *d*'s container document[p1039].
   2. If *parentDocument*'s origin is same origin[p940] with *d*'s origin and *parentDocument*'s character encoding is not UTF-16BE/LE, then return *parentDocument*'s character encoding, with the confidence[p1375] *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[p1375] *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[p1375] *tentative*. [UNIVCHARDET][p1587]

Note

User agents are generally discouraged from attempting to autodetect encodings for resources obtained over the network, since doing so involves inherently non-interoperable heuristics. Attempting to detect encodings based on an HTML document's preamble is especially tricky since HTML markup typically uses only ASCII characters, and HTML documents tend to begin with a lot of markup rather than with text content.

Note

The UTF-8 encoding has a highly detectable bit pattern. Files from the local file system that contain bytes with values greater than 0x7F which match the UTF-8 pattern are very likely to be UTF-8, while documents with byte sequences that do not match it are very likely not. When a user agent can examine the whole file, rather than just the preamble, detecting for UTF-8 specifically can be especially effective. [PPUTF8][p1585] [UTF8DET][p1587]

9. Otherwise, return an implementation-defined or user-specified default character encoding, with the confidence[p1375] *tentative*.

   In controlled environments or in environments where the encoding of documents can be prescribed (for example, for user agents intended for dedicated use in new networks), the comprehensive `UTF-8` encoding is suggested.

In other environments, the default encoding is typically dependent on the user's locale (an approximation of the languages, and thus often encodings, of the pages that the user is likely to frequent). The following table gives suggested defaults based on the user's locale, for compatibility with legacy content. Locales are identified by BCP 47 language tags. [BCP47]p1579 [ENCODING]p1582

Locale language		Suggested default encoding
ar	Arabic	windows-1256
az	Azeri	windows-1254
ba	Bashkir	windows-1251
be	Belarusian	windows-1251
bg	Bulgarian	windows-1251
cs	Czech	windows-1250
el	Greek	ISO-8859-7
et	Estonian	windows-1257
fa	Persian	windows-1256
he	Hebrew	windows-1255
hr	Croatian	windows-1250
hu	Hungarian	ISO-8859-2
ja	Japanese	Shift_JIS
kk	Kazakh	windows-1251
ko	Korean	EUC-KR
ku	Kurdish	windows-1254
ky	Kyrgyz	windows-1251
lt	Lithuanian	windows-1257
lv	Latvian	windows-1257
mk	Macedonian	windows-1251
pl	Polish	ISO-8859-2
ru	Russian	windows-1251
sah	Yakut	windows-1251
sk	Slovak	windows-1250
sl	Slovenian	ISO-8859-2
sr	Serbian	windows-1251
tg	Tajik	windows-1251
th	Thai	windows-874
tr	Turkish	windows-1254
tt	Tatar	windows-1251
uk	Ukrainian	windows-1251
vi	Vietnamese	windows-1258
zh-Hans, zh-CN, zh-SG	Chinese, Simplified	GBK
zh-Hant, zh-HK, zh-MO, zh-TW	Chinese, Traditional	Big5
All other locales		windows-1252

The contents of this table are derived from the intersection of Windows, Chrome, and Firefox defaults.

The document's character encoding must immediately be set to the value returned from this algorithm, at the same time as the user agent uses the returned value to select the decoder to use for the input byte stream.

When an algorithm requires a user agent to **prescan a byte stream to determine its encoding**, given some defined ***end condition***, then it must run the following steps. If at any point during these steps (including during instances of the get an attributep1379 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 encodingp1377 algorithm and return the result get an XML encodingp1380 applied to the same bytes that the prescan a byte stream to determine its encodingp1377 algorithm was applied to. Otherwise, these steps will return a character encoding.

1. Let *position* be a pointer to a byte in the input byte stream, initially pointing at the first byte.
2. Prescan for UTF-16 XML declarations: If *position* points to:

   ↪ A sequence of bytes starting with: 0x3C, 0x0, 0x3F, 0x0, 0x78, 0x0 (case-sensitive UTF-16 little-endian

'<?x')

Return UTF-16LE.

↪ **A sequence of bytes starting with: 0x0, 0x3C, 0x0, 0x3F, 0x0, 0x78 (case-sensitive UTF-16 big-endian '<?x')**

Return UTF-16BE.

Note

For historical reasons, the prefix is two bytes longer than in Appendix F of XML and the encoding name is not checked.

3. *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 '<!--' 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[p1379] and its value. If no attribute was sniffed, then jump to the *processing* step below.
7. If the attribute's name is already in *attribute list*, then return to the step labeled *attributes*.
8. Add the attribute's name to *attribute list*.
9. Run the appropriate step from the following list, if one applies:

↪ **If the attribute's name is "`http-equiv`"**

If the attribute's value is "`content-type`", then set *got pragma* to true.

↪ **If the attribute's name is "`content`"**

Apply the algorithm for extracting a character encoding from a `meta` element[p103], giving the attribute's value as the string to parse. If a character encoding is returned, and if *charset* is still set to null, let *charset* be the encoding returned, and set *need pragma* to true.

↪ **If the attribute's name is "`charset`"**

Let *charset* be the result of getting an encoding from the attribute's value, and set *need pragma* to false.

10. Return to the step labeled *attributes*.
11. *Processing*: If *need pragma* is null, then jump to the step below labeled *next byte*.
12. If *need pragma* is true but *got pragma* is false, then jump to the step below labeled *next byte*.
13. If *charset* is failure, then jump to the step below labeled *next byte*.
14. If *charset* is UTF-16BE/LE, then set *charset* to UTF-8.
15. If *charset* is x-user-defined, then set *charset* to windows-1252.
16. Return *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[p1379] 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.

4. *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[p1377] 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[p1379] 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[p1379] 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[p1379] 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[p1379] 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[p1379] 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*).

↪ **Anything else**

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.

When the prescan a byte stream to determine its encoding[p1377] algorithm is aborted without returning an encoding, **get an XML encoding** means doing this.

Note

Looking for syntax resembling an XML declaration, even in `text/html`[p1546]*, is necessary for compatibility with existing content.*

1. Let *encodingPosition* be a pointer to the start of the stream.

2. If *encodingPosition* does not point to the start of a byte sequence 0x3C, 0x3F, 0x78, 0x6D, 0x6C (`<?xml`), then return failure.

3. Let *xmlDeclarationEnd* be a pointer to the next byte in the input byte stream which is 0x3E (>). If there is no such byte, then return failure.

4. Set *encodingPosition* to the position of the first occurrence of the subsequence of bytes 0x65, 0x6E, 0x63, 0x6F, 0x64, 0x69, 0x6E, 0x67 (`encoding`) at or after the current *encodingPosition* and before *xmlDeclarationEnd*. If there is no such sequence, then return failure.

5. Advance *encodingPosition* past the 0x67 (g) byte.

6. While the byte at *encodingPosition* is less than or equal to 0x20 (i.e., it is either an ASCII space or control character), advance *encodingPosition* to the next byte.

7. If the byte at *encodingPosition* is not 0x3D (=), then return failure.

8. Advance *encodingPosition* to the next byte.

9. While the byte at *encodingPosition* is less than or equal to 0x20 (i.e., it is either an ASCII space or control character), advance *encodingPosition* to the next byte.

10. Let *quoteMark* be the byte at *encodingPosition*.

11. If *quoteMark* is not either 0x22 (") or 0x27 ('), then return failure.
12. Advance *encodingPosition* to the next byte.
13. Let *encodingEndPosition* be the position of the next occurrence of *quoteMark* at or after *encodingPosition*. If *quoteMark* does not occur again, then return failure.
14. Let *potentialEncoding* be the sequence of the bytes between *encodingPosition* (inclusive) and *encodingEndPosition* (exclusive).
15. If *potentialEncoding* contains one or more bytes whose byte value is 0x20 or below, then return failure.
16. Let *encoding* be the result of getting an encoding given *potentialEncoding* isomorphic decoded.
17. If the *encoding* is UTF-16BE/LE, then change it to UTF-8.
18. Return *encoding*.

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 §[p13 81]

User agents must support the encodings defined in *Encoding*, including, but not limited to, **UTF-8**, **ISO-8859-2**, **ISO-8859-7**, **ISO-8859-8**, **windows-874**, **windows-1250**, **windows-1251**, **windows-1252**, **windows-1254**, **windows-1255**, **windows-1256**, **windows-1257**, **windows-1258**, **GBK**, **Big5**, **ISO-2022-JP**, **Shift_JIS**, **EUC-KR**, **UTF-16BE**, **UTF-16LE**, **UTF-16BE/LE**, and **x-user-defined**. User agents must not support other encodings.

Note

The above prohibits supporting, for example, CESU-8, UTF-7, BOCU-1, SCSU, EBCDIC, and UTF-32. This specification does not make any attempt to support prohibited encodings in its algorithms; support and use of prohibited encodings would thus lead to unexpected behavior. [CESU8][p1579] [UTF7][p1587] [BOCU1][p1579] [SCSU][p1586]

13.2.3.4 Changing the encoding while parsing §[p13 81]

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[p1375] 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[p1375] 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[p1375] to *certain* and return. This happens when the encoding information found in the file matches what the encoding sniffing algorithm[p1375] 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[p1375] to *certain*, and return.
6. Otherwise, restart the navigate[p1063] algorithm, with *historyHandling*[p1063] set to "replace[p1062]" and other inputs kept the same, but this time skip the encoding sniffing algorithm[p1375] and instead just set the encoding to the new encoding and the confidence[p1375] 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[p1375] 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`[p196] element.

13.2.3.5 Preprocessing the input stream §[p13 82]

The **input stream** consists of the characters pushed into it as the input byte stream[p1374] is decoded or from the various APIs that directly manipulate the input stream.

Any occurrences of surrogates are surrogate-in-input-stream[p1373] parse errors[p1370]. Any occurrences of noncharacters are noncharacter-in-input-stream[p1373] parse errors[p1370] and any occurrences of controls other than ASCII whitespace and U+0000 NULL characters are control-character-in-input-stream[p1370] parse errors[p1370].

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[p1387] 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[p1387] stage.

The **next input character** is the first character in the input stream[p1382] that has not yet been **consumed** or explicitly ignored by the requirements in this section. Initially, the *next input character*[p1382] 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()`[p1223] 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[p1382]. If the parser is a script-created parser[p1221], then the end of the input stream[p1382] is reached when an **explicit "EOF" character** (inserted by the `document.close()`[p1222] 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 §[p13 82]

13.2.4.1 The insertion mode §[p13 82]

The **insertion mode** is a state variable that controls the primary operation of the tree construction stage.

Note

The rules for parsing tokens in foreign content[p1455] are not an insertion mode[p1382]. The tree construction dispatcher[p1417] selects them independently of the current insertion mode[p1382], which is left unchanged while they are in use.

Initially, the insertion mode[p1382] is "initial[p1424]". It can change to "before html[p1425]", "before head[p1426]", "in head[p1427]", "in head noscript[p1431]", "after head[p1431]", "in body[p1432]", "text[p1443]", "in table[p1445]", "in table text[p1447]", "in caption[p1447]", "in column group[p1448]", "in table body[p1449]", "in row[p1450]", "in cell[p1451]", "in template[p1452]", "after body[p1453]", "in frameset[p1453]", "after frameset[p1454]", "after after body[p1455]", and "after after frameset[p1455]" during the course of the parsing, as described in the tree construction[p1417] stage. The insertion mode affects how tokens are processed.

Several of these modes, namely "in head[p1427]", "in body[p1432]", and "in table[p1445]", 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[p1382]'s section, but must leave the insertion mode[p1382] unchanged unless the rules in *m* themselves switch the insertion mode[p1382] to a new value.

When the insertion mode is switched to "text[p1443]" or "in table text[p1447]", 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's fragment context element is non-null, then set *node* to that element (fragment case).
4. If *node* is a td or th element and *last* is false, then switch the insertion mode to "in cell" and return.
5. If *node* is a tr element, then switch the insertion mode to "in row" and return.
6. If *node* is a tbody, thead, or tfoot element, then switch the insertion mode to "in table body" and return.
7. If *node* is a caption element, then switch the insertion mode to "in caption" and return.
8. If *node* is a colgroup element, then switch the insertion mode to "in column group" and return.
9. If *node* is a table element, then switch the insertion mode to "in table" and return.
10. If *node* is a template element, then switch the insertion mode to the current template insertion mode and return.
11. If *node* is a head element and *last* is false, then switch the insertion mode to "in head" and return.
12. If *node* is a body element, then switch the insertion mode to "in body" and return.
13. If *node* is a frameset element, then switch the insertion mode to "in frameset" and return. (fragment case)
14. 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.
15. If *last* is true, then switch the insertion mode to "in body" and return. (fragment case)
16. Let *node* now be the node before *node* in the stack of open elements.
17. 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[p1458].

The **current node** is the bottommost node in this stack of open elements[p1383].

The **adjusted current node** is the parser's fragment context element[p1386] if that element is non-null and the stack of open elements[p1383] has only one element in it (fragment case[p1473]); otherwise, the adjusted current node[p1384] is the current node[p1384].

When the current node[p1384] is removed from the stack of open elements[p1383], process internal resource links[p331] given the current node[p1384]'s node document.

Elements in the stack of open elements[p1383] fall into the following categories:

Special

The following elements have varying levels of special parsing rules: HTML's address[p230], applet[p1530], area[p489], article[p214], aside[p222], base[p182], basefont[p1531], bgsound[p1530], blockquote[p244], body[p212], br[p310], button[p585], caption[p504], center[p1531], col[p506], colgroup[p505], dd[p258], details[p666], dir[p1530], div[p266], dl[p253], dt[p257], embed[p413], fieldset[p620], figcaption[p262], figure[p259], footer[p227], form[p532], frame[p1537], frameset[p1537], h1[p224], h2[p224], h3[p224], h4[p224], h5[p224], h6[p224], head[p180], header[p226], hgroup[p225], hr[p241], html[p179], iframe[p404], img[p359], input[p539], keygen[p1530], li[p251], link[p184], listing[p1530], main[p262], marquee[p1535], menu[p250], meta[p196], nav[p219], noembed[p1530], noframes[p1530], noscript[p703], object[p416], ol[p247], p[p238], param[p1530], plaintext[p1530], pre[p242], script[p685], search[p264], section[p216], select[p590], source[p355], style[p207], summary[p672], table[p496], tbody[p507], td[p512], template[p705], textarea[p606], tfoot[p509], th[p513], thead[p508], title[p181], tr[p510], track[p427], ul[p249], wbr[p311], xmp[p1531]; MathML mi, MathML mo, MathML mn, MathML ms, MathML mtext, and MathML annotation-xml; and SVG foreignObject, SVG desc, and SVG title.

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[p359] element.

Formatting

The following HTML elements are those that end up in the list of active formatting elements[p1385]: a[p267], b[p303], big[p1531], code[p297], em[p270], font[p1531], i[p302], nobr[p1531], s[p274], small[p272], strike[p1531], strong[p271], tt[p1531], and u[p304].

Ordinary

All other elements found while parsing an HTML document.

Note

Typically, the special[p1384] elements have the start and end tag tokens handled specifically, while ordinary[p1384] 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[p1383] is in the special[p1384] category. However, some elements (e.g., the option[p601] element) have their start or end tag tokens handled specifically, but are still not in the special[p1384] category, so that they get the ordinary[p1384] handling elsewhere.

The stack of open elements[p1383] 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[p1384] (the bottommost node of the stack).
2. If *node* is *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[p1383] 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[p179] element — is reached.)

The stack of open elements[p1383] is said to **have a particular element in scope** when it has that element in the specific scope[p1384] consisting of the following element types:

- applet[p1530]
- caption[p504]
- html[p179]
- table[p496]
- td[p512]
- th[p513]
- marquee[p1535]
- object[p416]

- select[p590]
- template[p705]
- MathML mi
- MathML mo
- MathML mn
- MathML ms
- MathML mtext
- MathML annotation-xml
- SVG foreignObject
- SVG desc
- SVG title

The stack of open elements[p1383] is said to **have a particular element in list item scope** when it has that element in the specific scope[p1384] consisting of the following element types:

- All the element types listed above for the *has an element in scope*[p1384] algorithm.
- ol[p247] in the HTML namespace
- ul[p249] in the HTML namespace

The stack of open elements[p1383] is said to **have a particular element in button scope** when it has that element in the specific scope[p1384] consisting of the following element types:

- All the element types listed above for the *has an element in scope*[p1384] algorithm.
- button[p585] in the HTML namespace

The stack of open elements[p1383] is said to **have a particular element in table scope** when it has that element in the specific scope[p1384] consisting of the following element types:

- html[p179] in the HTML namespace
- table[p496] in the HTML namespace
- template[p705] in the HTML namespace

Nothing happens if at any time any of the elements in the stack of open elements[p1383] are moved to a new location in, or removed from, the Document[p135] 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[p1442]), the stack is manipulated in a random-access fashion.

13.2.4.3 The list of active formatting elements §[p1385]

Initially, the **list of active formatting elements** is empty. It is used to handle mis-nested formatting element tags[p1384].

The list contains elements in the formatting[p1384] category, and markers[p1385]. The **markers** are inserted when entering applet[p1530], object[p416], marquee[p1535], template[p705], td[p512], th[p513], and caption[p504] elements, and are used to prevent formatting from "leaking" *into* applet[p1530], object[p416], marquee[p1535], template[p705], td[p512], th[p513], and caption[p504] elements.

In addition, each element in the list of active formatting elements[p1385] 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[p1385] after the last marker[p1385], if any, or anywhere in the list if there are no markers[p1385], that have the same tag name, namespace, and attributes as *element*, then remove the earliest such element from the list of active formatting elements[p1385]. For these purposes, the attributes must be compared as they were when the elements were created by the parser; two elements have the same attributes if all their parsed attributes can be paired such that the two attributes in each pair have identical names, namespaces, and values (the order of the attributes does not matter).

   Note

   This is the Noah's Ark clause. But with three per family instead of two.

2. Add *element* to the list of active formatting elements[p1385].

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[p1385], then there is nothing to reconstruct; stop this algorithm.
2. If the last (most recently added) entry in the list of active formatting elements[p1385] is a marker[p1385], or if it is an element that is in the stack of open elements[p1383], 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[p1385].
4. *Rewind*: If there are no entries before *entry* in the list of active formatting elements[p1385], then jump to the step labeled *create*.
5. Let *entry* be the entry one earlier than *entry* in the list of active formatting elements[p1385].
6. If *entry* is neither a marker[p1385] nor an element that is also in the stack of open elements[p1383], go to the step labeled *rewind*.
7. *Advance*: Let *entry* be the element one later than *entry* in the list of active formatting elements[p1385].
8. *Create*: Insert an HTML element[p1420] 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[p1385] is not the last entry in the list, return to the step labeled *advance*.

This has the effect of reopening all the formatting elements that were opened in the current body, cell, or caption (whichever is youngest) that haven't been explicitly closed.

Note

The way this specification is written, the list of active formatting elements[p1385] always consists of elements in chronological order with the least recently added element first and the most recently added element last (except for while steps 7 to 10 of the above algorithm are being executed, of course).

When the steps below require the 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[p1385].
2. Remove *entry* from the list of active formatting elements[p1385].
3. If *entry* was a marker[p1385], then stop the algorithm at this point. The list has been cleared up to the last marker[p1385].
4. Go to step 1.

13.2.4.4 The element pointers §p13 86

Initially, the **head element pointer** and the **form element pointer** are both null.

Once a `head`[p180] element has been parsed (whether implicitly or explicitly) the head element pointer[p1386] gets set to point to this node.

The `form` element pointer[p1386] points to the last `form`[p532] 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`[p705] elements.

An HTML parser[p1368] is **parsing template contents** if there is a `template`[p705] element on the stack of open elements[p1383], or the parser's fragment context element[p1386] is a `template`[p705] element.

13.2.4.5 Other parsing state flags §p13 86

The **root insertion target**, which is null or a `DocumentFragment` (initially null).

An `Element`-or-null **fragment context element**, initially null.

A boolean **allow declarative shadow roots** (initially false).

The **scripting mode**, which is a parser scripting mode[p1387]. It is initially set to Normal[p1387] if scripting was enabled[p1152] for the Document[p135] with which the parser is associated when the parser was created, and Disabled[p1387] otherwise.

A **parser scripting mode** is one of the following:

Normal

Scripts are processed when inserted, respecting async[p687] and defer[p687] attributes and blocking the parser when encountering a classic script[p1153].

Disabled

Scripts are disabled, and the noscript[p703] element can represent fallback content.

Inert

Scripts are enabled, however they are marked as already started[p692], essentially preventing them from executing. This is the default mode of the HTML fragment parsing algorithm[p1473].

Fragment

Scripts are executed as soon as they are inserted into the document as part of a the HTML fragment parsing algorithm[p1473], ignoring async[p687] and defer[p687] attributes. This mode is used by createContextualFragment()[p1232].

The **frameset-ok flag** is set to "ok" when the parser is created. It is set to "not ok" after certain tokens are seen, and back to "ok" when the body[p212] element is inserted implicitly.

13.2.5 Tokenization §[p1387]

Implementations must act as if they used the following state machine to tokenize HTML. The state machine must start in the data state[p1388]. Most states consume a single character, which may have various side-effects, and either switches the state machine to a new state to reconsume[p1387] the current input character[p1382], or switches it to a new state to consume the next character[p1382], 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[p1382], provide it with the current input character[p1382] instead.

The exact behavior of certain states depends on the insertion mode[p1382] and the stack of open elements[p1383]. Certain states also use a ***temporary buffer*** to track progress, and the character reference state[p1414] 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, a system identifier, and a ***force-quirks flag***. When a DOCTYPE token is created, its name, public identifier, and system identifier must be marked as missing (which is a distinct state from the empty string), and the *force-quirks flag*[p1387] 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*[p1387] 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[p1417] 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[p685] element can result in scripts executing and using the dynamic markup insertion[p1220] APIs to insert characters into the stream being tokenized.)

Note

Creating a token and emitting it are distinct actions. It is possible for a token to be created but implicitly abandoned (never emitted), e.g. if the file ends unexpectedly while processing the characters that are being parsed into a start tag token.

When a start tag token is emitted with its *self-closing flag*[p1387] 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[p1373] parse error[p1370].

When an end tag token is emitted with attributes, that is an end-tag-with-attributes[p1371] parse error[p1370].

When an end tag token is emitted with its *self-closing flag*[p1387] set, that is an end-tag-with-trailing-solidus[p1371] parse error[p1370].

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[p1366] is said to be **consumed as part of an attribute** if the *return state*[p1387] is either attribute value (double-quoted) state[p1399], attribute value (single-quoted) state[p1400], or attribute value (unquoted) state[p1400].

When a state says to **flush code points consumed as a character reference**, it means that for each code point in the *temporary buffer*[p1387] (in the order they were added to the buffer), the 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[p1388], or emit the code point as a character token otherwise.

To **convert the temporary buffer to a comment**, create a comment token whose data is the concatenation of "?" and the code points in the *temporary buffer*[p1387], in the order they were added to the buffer.

Note

This is used when a processing instruction is found to have an invalid target and is instead treated as a bogus comment.

Before each step of the tokenizer, the user agent must first check the parser pause flag[p1370]. 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 §[p13 88]

Consume the next input character[p1382]:

↪ **U+0026 AMPERSAND (&)**
Set the *return state*[p1387] to the data state[p1388]. Switch to the character reference state[p1414].

↪ **U+003C LESS-THAN SIGN (<)**
Switch to the tag open state[p1389].

↪ **U+0000 NULL**
This is an unexpected-null-character[p1374] parse error[p1370]. Emit the current input character[p1382] as a character token.

↪ **EOF**
Emit an end-of-file token.

↪ **Anything else**
Emit the current input character[p1382] as a character token.

13.2.5.2 RCDATA state §[p13 88]

Consume the next input character[p1382]:

↪ **U+0026 AMPERSAND (&)**
Set the *return state*[p1387] to the RCDATA state[p1388]. Switch to the character reference state[p1414].

↪ **U+003C LESS-THAN SIGN (<)**
Switch to the RCDATA less-than sign state[p1390].

↪ **U+0000 NULL**
This is an unexpected-null-character[p1374] parse error[p1370]. Emit a U+FFFD REPLACEMENT CHARACTER character token.

↪ **EOF**
Emit an end-of-file token.

↪ **Anything else**
Emit the current input character[p1382] as a character token.

13.2.5.3 RAWTEXT state §[p13 89]

Consume the next input character[p1382]:

↪ **U+003C LESS-THAN SIGN (<)**
Switch to the RAWTEXT less-than sign state[p1391].

↪ **U+0000 NULL**
This is an unexpected-null-character[p1374] parse error[p1370]. Emit a U+FFFD REPLACEMENT CHARACTER character token.

↪ **EOF**
Emit an end-of-file token.

↪ **Anything else**
Emit the current input character[p1382] as a character token.

13.2.5.4 Script data state §[p13 89]

Consume the next input character[p1382]:

↪ **U+003C LESS-THAN SIGN (<)**
Switch to the script data less-than sign state[p1392].

↪ **U+0000 NULL**
This is an unexpected-null-character[p1374] parse error[p1370]. Emit a U+FFFD REPLACEMENT CHARACTER character token.

↪ **EOF**
Emit an end-of-file token.

↪ **Anything else**
Emit the current input character[p1382] as a character token.

13.2.5.5 PLAINTEXT state §[p13 89]

Consume the next input character[p1382]:

↪ **U+0000 NULL**
This is an unexpected-null-character[p1374] parse error[p1370]. Emit a U+FFFD REPLACEMENT CHARACTER character token.

↪ **EOF**
Emit an end-of-file token.

↪ **Anything else**
Emit the current input character[p1382] as a character token.

13.2.5.6 Tag open state §[p13 89]

Consume the next input character[p1382]:

↪ **U+0021 EXCLAMATION MARK (!)**
Switch to the markup declaration open state[p1402].

↪ **U+002F SOLIDUS (/)**
Switch to the end tag open state[p1390].

↪ **ASCII alpha**
Create a new start tag token, set its tag name to the empty string. Reconsume[p1387] in the tag name state[p1390].

↪ **U+003F QUESTION MARK (?)**
Set the *temporary buffer*[p1387] to the empty string. Switch to the processing instruction open state[p1412].

↪ **EOF**

This is an eof-before-tag-name[p1371] parse error[p1370]. Emit a U+003C LESS-THAN SIGN character token and an end-of-file token.

↪ **Anything else**

This is an invalid-first-character-of-tag-name[p1371] parse error[p1370]. Emit a U+003C LESS-THAN SIGN character token. Reconsume[p1387] in the data state[p1388].

13.2.5.7 End tag open state §[p1390]

Consume the next input character[p1382]:

↪ **ASCII alpha**

Create a new end tag token, set its tag name to the empty string. Reconsume[p1387] in the tag name state[p1390].

↪ **U+003E GREATER-THAN SIGN (>)**

This is a missing-end-tag-name[p1372] parse error[p1370]. Switch to the data state[p1388].

↪ **EOF**

This is an eof-before-tag-name[p1371] parse error[p1370]. 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[p1371] parse error[p1370]. Create a comment token whose data is the empty string. Reconsume[p1387] in the bogus comment state[p1401].

13.2.5.8 Tag name state §[p1390]

Consume the next input character[p1382]:

↪ **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[p1398].

↪ **U+002F SOLIDUS (/)**

Switch to the self-closing start tag state[p1401].

↪ **U+003E GREATER-THAN SIGN (>)**

Switch to the data state[p1388]. Emit the current tag token.

↪ **ASCII upper alpha**

Append the lowercase version of the current input character[p1382] (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[p1374] parse error[p1370]. Append a U+FFFD REPLACEMENT CHARACTER character to the current tag token's tag name.

↪ **EOF**

This is an eof-in-tag[p1371] parse error[p1370]. Emit an end-of-file token.

↪ **Anything else**

Append the current input character[p1382] to the current tag token's tag name.

13.2.5.9 RCDATA less-than sign state §[p1390]

Consume the next input character[p1382]:

↪ **U+002F SOLIDUS (/)**

Set the *temporary buffer*[p1387] to the empty string. Switch to the RCDATA end tag open state[p1391].

↪ **Anything else**

Emit a U+003C LESS-THAN SIGN character token. Reconsume[p1387] in the RCDATA state[p1388].

13.2.5.10 RCDATA end tag open state §[p13 91]

Consume the next input character[p1382]:

↪ **ASCII alpha**

Create a new end tag token, set its tag name to the empty string. Reconsume[p1387] in the RCDATA end tag name state[p1391].

↪ **Anything else**

Emit a U+003C LESS-THAN SIGN character token and a U+002F SOLIDUS character token. Reconsume[p1387] in the RCDATA state[p1388].

13.2.5.11 RCDATA end tag name state §[p13 91]

Consume the next input character[p1382]:

↪ **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[p1388], then switch to the before attribute name state[p1398]. 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[p1388], then switch to the self-closing start tag state[p1401]. 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[p1388], then switch to the data state[p1388] 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[p1382] (add 0x0020 to the character's code point) to the current tag token's tag name. Append the current input character[p1382] to the *temporary buffer*[p1387].

↪ **ASCII lower alpha**

Append the current input character[p1382] to the current tag token's tag name. Append the current input character[p1382] to the *temporary buffer*[p1387].

↪ **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*[p1387] (in the order they were added to the buffer). Reconsume[p1387] in the RCDATA state[p1388].

13.2.5.12 RAWTEXT less-than sign state §[p13 91]

Consume the next input character[p1382]:

↪ **U+002F SOLIDUS (/)**

Set the *temporary buffer*[p1387] to the empty string. Switch to the RAWTEXT end tag open state[p1392].

↪ **Anything else**

Emit a U+003C LESS-THAN SIGN character token. Reconsume[p1387] in the RAWTEXT state[p1389].

13.2.5.13 RAWTEXT end tag open state §[p1392]

Consume the next input character[p1382]:

↪ **ASCII alpha**

Create a new end tag token, set its tag name to the empty string. Reconsume[p1387] in the RAWTEXT end tag name state[p1392].

↪ **Anything else**

Emit a U+003C LESS-THAN SIGN character token and a U+002F SOLIDUS character token. Reconsume[p1387] in the RAWTEXT state[p1389].

13.2.5.14 RAWTEXT end tag name state §[p1392]

Consume the next input character[p1382]:

↪ **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[p1388], then switch to the before attribute name state[p1398]. 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[p1388], then switch to the self-closing start tag state[p1401]. 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[p1388], then switch to the data state[p1388] 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[p1382] (add 0x0020 to the character's code point) to the current tag token's tag name. Append the current input character[p1382] to the *temporary buffer*[p1387].

↪ **ASCII lower alpha**

Append the current input character[p1382] to the current tag token's tag name. Append the current input character[p1382] to the *temporary buffer*[p1387].

↪ **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*[p1387] (in the order they were added to the buffer). Reconsume[p1387] in the RAWTEXT state[p1389].

13.2.5.15 Script data less-than sign state §[p1392]

Consume the next input character[p1382]:

↪ **U+002F SOLIDUS (/)**

Set the *temporary buffer*[p1387] to the empty string. Switch to the script data end tag open state[p1392].

↪ **U+0021 EXCLAMATION MARK (!)**

Switch to the script data escape start state[p1393]. 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[p1387] in the script data state[p1389].

13.2.5.16 Script data end tag open state §[p1392]

Consume the next input character[p1382]:

↪ **ASCII alpha**

Create a new end tag token, set its tag name to the empty string. Reconsume[p1387] in the script data end tag name state[p1393].

↪ **Anything else**

Emit a U+003C LESS-THAN SIGN character token and a U+002F SOLIDUS character token. Reconsume[p1387] in the script data state[p1389].

13.2.5.17 Script data end tag name state §[p1393]

Consume the next input character[p1382]:

↪ **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[p1388], then switch to the before attribute name state[p1398]. 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[p1388], then switch to the self-closing start tag state[p1401]. 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[p1388], then switch to the data state[p1388] 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[p1382] (add 0x0020 to the character's code point) to the current tag token's tag name. Append the current input character[p1382] to the *temporary buffer*[p1387].

↪ **ASCII lower alpha**

Append the current input character[p1382] to the current tag token's tag name. Append the current input character[p1382] to the *temporary buffer*[p1387].

↪ **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*[p1387] (in the order they were added to the buffer). Reconsume[p1387] in the script data state[p1389].

13.2.5.18 Script data escape start state §[p1393]

Consume the next input character[p1382]:

↪ **U+002D HYPHEN-MINUS (-)**

Switch to the script data escape start dash state[p1393]. Emit a U+002D HYPHEN-MINUS character token.

↪ **Anything else**

Reconsume[p1387] in the script data state[p1389].

13.2.5.19 Script data escape start dash state §[p1393]

Consume the next input character[p1382]:

↪ **U+002D HYPHEN-MINUS (-)**

Switch to the script data escaped dash dash state[p1394]. Emit a U+002D HYPHEN-MINUS character token.

↪ **Anything else**

Reconsume[p1387] in the script data state[p1389].

13.2.5.20 Script data escaped state §[p13 94]

Consume the next input character[p1382]:

↪ **U+002D HYPHEN-MINUS (-)**
Switch to the script data escaped dash state[p1394]. Emit a U+002D HYPHEN-MINUS character token.

↪ **U+003C LESS-THAN SIGN (<)**
Switch to the script data escaped less-than sign state[p1395].

↪ **U+0000 NULL**
This is an unexpected-null-character[p1374] parse error[p1370]. Emit a U+FFFD REPLACEMENT CHARACTER character token.

↪ **EOF**
This is an eof-in-script-html-comment-like-text[p1371] parse error[p1370]. Emit an end-of-file token.

↪ **Anything else**
Emit the current input character[p1382] as a character token.

13.2.5.21 Script data escaped dash state §[p13 94]

Consume the next input character[p1382]:

↪ **U+002D HYPHEN-MINUS (-)**
Switch to the script data escaped dash dash state[p1394]. Emit a U+002D HYPHEN-MINUS character token.

↪ **U+003C LESS-THAN SIGN (<)**
Switch to the script data escaped less-than sign state[p1395].

↪ **U+0000 NULL**
This is an unexpected-null-character[p1374] parse error[p1370]. Switch to the script data escaped state[p1394]. Emit a U+FFFD REPLACEMENT CHARACTER character token.

↪ **EOF**
This is an eof-in-script-html-comment-like-text[p1371] parse error[p1370]. Emit an end-of-file token.

↪ **Anything else**
Switch to the script data escaped state[p1394]. Emit the current input character[p1382] as a character token.

13.2.5.22 Script data escaped dash dash state §[p13 94]

Consume the next input character[p1382]:

↪ **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[p1395].

↪ **U+003E GREATER-THAN SIGN (>)**
Switch to the script data state[p1389]. Emit a U+003E GREATER-THAN SIGN character token.

↪ **U+0000 NULL**
This is an unexpected-null-character[p1374] parse error[p1370]. Switch to the script data escaped state[p1394]. Emit a U+FFFD REPLACEMENT CHARACTER character token.

↪ **EOF**
This is an eof-in-script-html-comment-like-text[p1371] parse error[p1370]. Emit an end-of-file token.

↪ **Anything else**
Switch to the script data escaped state[p1394]. Emit the current input character[p1382] as a character token.

13.2.5.23 Script data escaped less-than sign state §[p1395]

Consume the next input character[p1382]:

↪ **U+002F SOLIDUS (/)**

Set the *temporary buffer*[p1387] to the empty string. Switch to the script data escaped end tag open state[p1395].

↪ **ASCII alpha**

Set the *temporary buffer*[p1387] to the empty string. Emit a U+003C LESS-THAN SIGN character token. Reconsume[p1387] in the script data double escape start state[p1396].

↪ **Anything else**

Emit a U+003C LESS-THAN SIGN character token. Reconsume[p1387] in the script data escaped state[p1394].

13.2.5.24 Script data escaped end tag open state §[p1395]

Consume the next input character[p1382]:

↪ **ASCII alpha**

Create a new end tag token, set its tag name to the empty string. Reconsume[p1387] in the script data escaped end tag name state[p1395].

↪ **Anything else**

Emit a U+003C LESS-THAN SIGN character token and a U+002F SOLIDUS character token. Reconsume[p1387] in the script data escaped state[p1394].

13.2.5.25 Script data escaped end tag name state §[p1395]

Consume the next input character[p1382]:

↪ **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[p1388], then switch to the before attribute name state[p1398]. 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[p1388], then switch to the self-closing start tag state[p1401]. 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[p1388], then switch to the data state[p1388] 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[p1382] (add 0x0020 to the character's code point) to the current tag token's tag name. Append the current input character[p1382] to the *temporary buffer*[p1387].

↪ **ASCII lower alpha**

Append the current input character[p1382] to the current tag token's tag name. Append the current input character[p1382] to the *temporary buffer*[p1387].

↪ **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*[p1387] (in the order they were added to the buffer). Reconsume[p1387] in the script data escaped state[p1394].

13.2.5.26 Script data double escape start state §[p13 96]

Consume the next input character[p1382]:

↪ **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*[p1387] is "`script`", then switch to the script data double escaped state[p1396]. Otherwise, switch to the script data escaped state[p1394]. Emit the current input character[p1382] as a character token.

↪ **ASCII upper alpha**

Append the lowercase version of the current input character[p1382] (add 0x0020 to the character's code point) to the *temporary buffer*[p1387]. Emit the current input character[p1382] as a character token.

↪ **ASCII lower alpha**

Append the current input character[p1382] to the *temporary buffer*[p1387]. Emit the current input character[p1382] as a character token.

↪ **Anything else**

Reconsume[p1387] in the script data escaped state[p1394].

13.2.5.27 Script data double escaped state §[p13 96]

Consume the next input character[p1382]:

↪ **U+002D HYPHEN-MINUS (-)**

Switch to the script data double escaped dash state[p1396]. Emit a U+002D HYPHEN-MINUS character token.

↪ **U+003C LESS-THAN SIGN (<)**

Switch to the script data double escaped less-than sign state[p1397]. Emit a U+003C LESS-THAN SIGN character token.

↪ **U+0000 NULL**

This is an unexpected-null-character[p1374] parse error[p1370]. Emit a U+FFFD REPLACEMENT CHARACTER character token.

↪ **EOF**

This is an eof-in-script-html-comment-like-text[p1371] parse error[p1370]. Emit an end-of-file token.

↪ **Anything else**

Emit the current input character[p1382] as a character token.

13.2.5.28 Script data double escaped dash state §[p13 96]

Consume the next input character[p1382]:

↪ **U+002D HYPHEN-MINUS (-)**

Switch to the script data double escaped dash dash state[p1397]. Emit a U+002D HYPHEN-MINUS character token.

↪ **U+003C LESS-THAN SIGN (<)**

Switch to the script data double escaped less-than sign state[p1397]. Emit a U+003C LESS-THAN SIGN character token.

↪ **U+0000 NULL**

This is an unexpected-null-character[p1374] parse error[p1370]. Switch to the script data double escaped state[p1396]. Emit a U+FFFD REPLACEMENT CHARACTER character token.

↪ **EOF**

This is an eof-in-script-html-comment-like-text[p1371] parse error[p1370]. Emit an end-of-file token.

↪ **Anything else**

Switch to the script data double escaped state[p1396]. Emit the current input character[p1382] as a character token.

13.2.5.29 Script data double escaped dash dash state §p13 97

Consume the next input character[p1382]:

↪ **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[p1397]. Emit a U+003C LESS-THAN SIGN character token.

↪ **U+003E GREATER-THAN SIGN (>)**

Switch to the script data state[p1389]. Emit a U+003E GREATER-THAN SIGN character token.

↪ **U+0000 NULL**

This is an unexpected-null-character[p1374] parse error[p1370]. Switch to the script data double escaped state[p1396]. Emit a U+FFFD REPLACEMENT CHARACTER character token.

↪ **EOF**

This is an eof-in-script-html-comment-like-text[p1371] parse error[p1370]. Emit an end-of-file token.

↪ **Anything else**

Switch to the script data double escaped state[p1396]. Emit the current input character[p1382] as a character token.

13.2.5.30 Script data double escaped less-than sign state §p13 97

Consume the next input character[p1382]:

↪ **U+002F SOLIDUS (/)**

Set the *temporary buffer*[p1387] to the empty string. Switch to the script data double escape end state[p1397]. Emit a U+002F SOLIDUS character token.

↪ **Anything else**

Reconsume[p1387] in the script data double escaped state[p1396].

13.2.5.31 Script data double escape end state §p13 97

Consume the next input character[p1382]:

↪ **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*[p1387] is "`script`", then switch to the script data escaped state[p1394]. Otherwise, switch to the script data double escaped state[p1396]. Emit the current input character[p1382] as a character token.

↪ **ASCII upper alpha**

Append the lowercase version of the current input character[p1382] (add 0x0020 to the character's code point) to the *temporary buffer*[p1387]. Emit the current input character[p1382] as a character token.

↪ **ASCII lower alpha**

Append the current input character[p1382] to the *temporary buffer*[p1387]. Emit the current input character[p1382] as a character token.

↪ **Anything else**

Reconsume[p1387] in the script data double escaped state[p1396].

13.2.5.32 Before attribute name state §[p1398]

Consume the next input character[p1382]:

↪ **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-THAN SIGN (>)**
↪ **EOF**

Reconsume[p1387] in the after attribute name state[p1399].

↪ **U+003D EQUALS SIGN (=)**

This is an unexpected-equals-sign-before-attribute-name[p1374] parse error[p1370]. Start a new attribute in the current tag token. Set that attribute's name to the current input character[p1382], and its value to the empty string. Switch to the attribute name state[p1398].

↪ **Anything else**

Start a new attribute in the current tag token. Set that attribute name and value to the empty string. Reconsume[p1387] in the attribute name state[p1398].

13.2.5.33 Attribute name state §[p1398]

Consume the next input character[p1382]:

↪ **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 (>)**
↪ **EOF**

Reconsume[p1387] in the after attribute name state[p1399].

↪ **U+003D EQUALS SIGN (=)**

Switch to the before attribute value state[p1399].

↪ **ASCII upper alpha**

Append the lowercase version of the current input character[p1382] (add 0x0020 to the character's code point) to the current attribute's name.

↪ **U+0000 NULL**

This is an unexpected-null-character[p1374] parse error[p1370]. Append a U+FFFD REPLACEMENT CHARACTER character to the current attribute's name.

↪ **U+0022 QUOTATION MARK (")**
↪ **U+0027 APOSTROPHE (')**
↪ **U+003C LESS-THAN SIGN (<)**

This is an unexpected-character-in-attribute-name[p1373] parse error[p1370]. Treat it as per the "anything else" entry below.

↪ **Anything else**

Append the current input character[p1382] 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[p1371] parse error[p1370] and the new attribute must be removed from the token.

Note

If an attribute is so removed from a token, it, and the value that gets associated with it, if any, are never subsequently used by the parser, and are therefore effectively discarded. Removing the attribute in this way does not change its status as the "current attribute" for the purposes of the tokenizer, however.

13.2.5.34 After attribute name state §[p13 99]

Consume the next input character[p1382]:

- ↪ **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[p1401].

- ↪ **U+003D EQUALS SIGN (=)**

  Switch to the before attribute value state[p1399].

- ↪ **U+003E GREATER-THAN SIGN (>)**

  Switch to the data state[p1388]. Emit the current tag token.

- ↪ **EOF**

  This is an eof-in-tag[p1371] parse error[p1370]. 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. Reconsume[p1387] in the attribute name state[p1398].

13.2.5.35 Before attribute value state §[p13 99]

Consume the next input character[p1382]:

- ↪ **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[p1399].

- ↪ **U+0027 APOSTROPHE (')**

  Switch to the attribute value (single-quoted) state[p1400].

- ↪ **U+003E GREATER-THAN SIGN (>)**

  This is a missing-attribute-value[p1372] parse error[p1370]. Switch to the data state[p1388]. Emit the current tag token.

- ↪ **Anything else**

  Reconsume[p1387] in the attribute value (unquoted) state[p1400].

13.2.5.36 Attribute value (double-quoted) state §[p13 99]

Consume the next input character[p1382]:

- ↪ **U+0022 QUOTATION MARK (")**

  Switch to the after attribute value (quoted) state[p1401].

↪ **U+0026 AMPERSAND (&)**

Set the *return state*[p1387] to the attribute value (double-quoted) state[p1399]. Switch to the character reference state[p1414].

↪ **U+0000 NULL**

This is an unexpected-null-character[p1374] parse error[p1370]. Append a U+FFFD REPLACEMENT CHARACTER character to the current attribute's value.

↪ **EOF**

This is an eof-in-tag[p1371] parse error[p1370]. Emit an end-of-file token.

↪ **Anything else**

Append the current input character[p1382] to the current attribute's value.

13.2.5.37 Attribute value (single-quoted) state §[p14]

Consume the next input character[p1382]:

↪ **U+0027 APOSTROPHE (')**

Switch to the after attribute value (quoted) state[p1401].

↪ **U+0026 AMPERSAND (&)**

Set the *return state*[p1387] to the attribute value (single-quoted) state[p1400]. Switch to the character reference state[p1414].

↪ **U+0000 NULL**

This is an unexpected-null-character[p1374] parse error[p1370]. Append a U+FFFD REPLACEMENT CHARACTER character to the current attribute's value.

↪ **EOF**

This is an eof-in-tag[p1371] parse error[p1370]. Emit an end-of-file token.

↪ **Anything else**

Append the current input character[p1382] to the current attribute's value.

13.2.5.38 Attribute value (unquoted) state §[p14]

Consume the next input character[p1382]:

↪ **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[p1398].

↪ **U+0026 AMPERSAND (&)**

Set the *return state*[p1387] to the attribute value (unquoted) state[p1400]. Switch to the character reference state[p1414].

↪ **U+003E GREATER-THAN SIGN (>)**

Switch to the data state[p1388]. Emit the current tag token.

↪ **U+0000 NULL**

This is an unexpected-null-character[p1374] parse error[p1370]. Append a U+FFFD REPLACEMENT CHARACTER character to the current attribute's value.

↪ **U+0022 QUOTATION MARK (")**
↪ **U+0027 APOSTROPHE (')**
↪ **U+003C LESS-THAN SIGN (<)**
↪ **U+003D EQUALS SIGN (=)**
↪ **U+0060 GRAVE ACCENT (`)**

This is an unexpected-character-in-unquoted-attribute-value[p1374] parse error[p1370]. Treat it as per the "anything else" entry below.

↪ **EOF**

This is an eof-in-tag[p1371] parse error[p1370]. Emit an end-of-file token.

↪ **Anything else**

Append the current input character[p1382] to the current attribute's value.

13.2.5.39 After attribute value (quoted) state §[p14 01]

Consume the next input character[p1382]:

↪ **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[p1398].

↪ **U+002F SOLIDUS (/)**

Switch to the self-closing start tag state[p1401].

↪ **U+003E GREATER-THAN SIGN (>)**

Switch to the data state[p1388]. Emit the current tag token.

↪ **EOF**

This is an eof-in-tag[p1371] parse error[p1370]. Emit an end-of-file token.

↪ **Anything else**

This is a missing-whitespace-between-attributes[p1373] parse error[p1370]. Reconsume[p1387] in the before attribute name state[p1398].

13.2.5.40 Self-closing start tag state §[p14 01]

Consume the next input character[p1382]:

↪ **U+003E GREATER-THAN SIGN (>)**

Set the *self-closing flag*[p1387] of the current tag token. Switch to the data state[p1388]. Emit the current tag token.

↪ **EOF**

This is an eof-in-tag[p1371] parse error[p1370]. Emit an end-of-file token.

↪ **Anything else**

This is an unexpected-solidus-in-tag[p1374] parse error[p1370]. Reconsume[p1387] in the before attribute name state[p1398].

13.2.5.41 Bogus comment state §[p14 01]

Consume the next input character[p1382]:

↪ **U+003E GREATER-THAN SIGN (>)**

Switch to the data state[p1388]. Emit the current comment token.

↪ **EOF**

Emit the current comment token. Emit an end-of-file token.

↪ **U+0000 NULL**

This is an unexpected-null-character[p1374] parse error[p1370]. Append a U+FFFD REPLACEMENT CHARACTER character to the comment token's data.

↪ **Anything else**

Append the current input character[p1382] to the comment token's data.

13.2.5.42 Markup declaration open state §[p1402]

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[p1402].
- **ASCII case-insensitive match for "DOCTYPE"**
  Consume those characters and switch to the DOCTYPE state[p1404].
- **"[CDATA["**
  Consume those characters. If there is an adjusted current node[p1384] and it is not an element in the HTML namespace, then switch to the CDATA section state[p1411]. Otherwise, this is a cdata-in-html-content[p1370] parse error[p1370]. Create a comment token whose data is "[CDATA[". Switch to the bogus comment state[p1401].
- **Anything else**
  This is an incorrectly-opened-comment[p1371] parse error[p1370]. Create a comment token whose data is the empty string. Switch to the bogus comment state[p1401] (don't consume anything in the current state).

13.2.5.43 Comment start state §[p1402]

Consume the next input character[p1382]:

- **U+002D HYPHEN-MINUS (-)**
  Switch to the comment start dash state[p1402].
- **U+003E GREATER-THAN SIGN (>)**
  This is an abrupt-closing-of-empty-comment[p1370] parse error[p1370]. Switch to the data state[p1388]. Emit the current comment token.
- **Anything else**
  Reconsume[p1387] in the comment state[p1402].

13.2.5.44 Comment start dash state §[p1402]

Consume the next input character[p1382]:

- **U+002D HYPHEN-MINUS (-)**
  Switch to the comment end state[p1404].
- **U+003E GREATER-THAN SIGN (>)**
  This is an abrupt-closing-of-empty-comment[p1370] parse error[p1370]. Switch to the data state[p1388]. Emit the current comment token.
- **EOF**
  This is an eof-in-comment[p1371] parse error[p1370]. Emit the current comment token. Emit an end-of-file token.
- **Anything else**
  Append a U+002D HYPHEN-MINUS character (-) to the comment token's data. Reconsume[p1387] in the comment state[p1402].

13.2.5.45 Comment state §[p1402]

Consume the next input character[p1382]:

- **U+003C LESS-THAN SIGN (<)**
  Append the current input character[p1382] to the comment token's data. Switch to the comment less-than sign state[p1403].
- **U+002D HYPHEN-MINUS (-)**
  Switch to the comment end dash state[p1404].

↪ **U+0000 NULL**

This is an unexpected-null-character[p1374] parse error[p1370]. Append a U+FFFD REPLACEMENT CHARACTER character to the comment token's data.

↪ **EOF**

This is an eof-in-comment[p1371] parse error[p1370]. Emit the current comment token. Emit an end-of-file token.

↪ **Anything else**

Append the current input character[p1382] to the comment token's data.

13.2.5.46 Comment less-than sign state §[p14 03]

Consume the next input character[p1382]:

↪ **U+0021 EXCLAMATION MARK (!)**

Append the current input character[p1382] to the comment token's data. Switch to the comment less-than sign bang state[p1403].

↪ **U+003C LESS-THAN SIGN (<)**

Append the current input character[p1382] to the comment token's data.

↪ **Anything else**

Reconsume[p1387] in the comment state[p1402].

13.2.5.47 Comment less-than sign bang state §[p14 03]

Consume the next input character[p1382]:

↪ **U+002D HYPHEN-MINUS (-)**

Switch to the comment less-than sign bang dash state[p1403].

↪ **Anything else**

Reconsume[p1387] in the comment state[p1402].

13.2.5.48 Comment less-than sign bang dash state §[p14 03]

Consume the next input character[p1382]:

↪ **U+002D HYPHEN-MINUS (-)**

Switch to the comment less-than sign bang dash dash state[p1403].

↪ **Anything else**

Reconsume[p1387] in the comment end dash state[p1404].

13.2.5.49 Comment less-than sign bang dash dash state §[p14 03]

Consume the next input character[p1382]:

↪ **U+003E GREATER-THAN SIGN (>)**

↪ **EOF**

Reconsume[p1387] in the comment end state[p1404].

↪ **Anything else**

This is a nested-comment[p1373] parse error[p1370]. Reconsume[p1387] in the comment end state[p1404].

13.2.5.50 Comment end dash state §[p1404]

Consume the next input character[p1382]:

↪ **U+002D HYPHEN-MINUS (-)**
Switch to the comment end state[p1404].

↪ **EOF**
This is an eof-in-comment[p1371] parse error[p1370]. Emit the current comment token. Emit an end-of-file token.

↪ **Anything else**
Append a U+002D HYPHEN-MINUS character (-) to the comment token's data. Reconsume[p1387] in the comment state[p1402].

13.2.5.51 Comment end state §[p1404]

Consume the next input character[p1382]:

↪ **U+003E GREATER-THAN SIGN (>)**
Switch to the data state[p1388]. Emit the current comment token.

↪ **U+0021 EXCLAMATION MARK (!)**
Switch to the comment end bang state[p1404].

↪ **U+002D HYPHEN-MINUS (-)**
Append a U+002D HYPHEN-MINUS character (-) to the comment token's data.

↪ **EOF**
This is an eof-in-comment[p1371] parse error[p1370]. Emit the current comment token. Emit an end-of-file token.

↪ **Anything else**
Append two U+002D HYPHEN-MINUS characters (-) to the comment token's data. Reconsume[p1387] in the comment state[p1402].

13.2.5.52 Comment end bang state §[p1404]

Consume the next input character[p1382]:

↪ **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[p1404].

↪ **U+003E GREATER-THAN SIGN (>)**
This is an incorrectly-closed-comment[p1371] parse error[p1370]. Switch to the data state[p1388]. Emit the current comment token.

↪ **EOF**
This is an eof-in-comment[p1371] parse error[p1370]. Emit the current 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[p1387] in the comment state[p1402].

13.2.5.53 DOCTYPE state §[p1404]

Consume the next input character[p1382]:

↪ **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[p1405].

↪ **U+003E GREATER-THAN SIGN (>)**

Reconsume[p1387] in the before DOCTYPE name state[p1405].

↪ **EOF**

This is an eof-in-doctype[p1371] parse error[p1370]. Create a new DOCTYPE token. Set its *force-quirks flag*[p1387] to *on*. Emit the current token. Emit an end-of-file token.

↪ **Anything else**

This is a missing-whitespace-before-doctype-name[p1372] parse error[p1370]. Reconsume[p1387] in the before DOCTYPE name state[p1405].

13.2.5.54 Before DOCTYPE name state §[p1405]

Consume the next input character[p1382]:

↪ **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[p1382] (add 0x0020 to the character's code point). Switch to the DOCTYPE name state[p1405].

↪ **U+0000 NULL**

This is an unexpected-null-character[p1374] parse error[p1370]. Create a new DOCTYPE token. Set the token's name to a U+FFFD REPLACEMENT CHARACTER character. Switch to the DOCTYPE name state[p1405].

↪ **U+003E GREATER-THAN SIGN (>)**

This is a missing-doctype-name[p1372] parse error[p1370]. Create a new DOCTYPE token. Set its *force-quirks flag*[p1387] to *on*. Switch to the data state[p1388]. Emit the current token.

↪ **EOF**

This is an eof-in-doctype[p1371] parse error[p1370]. Create a new DOCTYPE token. Set its *force-quirks flag*[p1387] to *on*. Emit the current token. Emit an end-of-file token.

↪ **Anything else**

Create a new DOCTYPE token. Set the token's name to the current input character[p1382]. Switch to the DOCTYPE name state[p1405].

13.2.5.55 DOCTYPE name state §[p1405]

Consume the next input character[p1382]:

↪ **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[p1406].

↪ **U+003E GREATER-THAN SIGN (>)**

Switch to the data state[p1388]. Emit the current DOCTYPE token.

↪ **ASCII upper alpha**

Append the lowercase version of the current input character[p1382] (add 0x0020 to the character's code point) to the current DOCTYPE token's name.

↪ **U+0000 NULL**

This is an unexpected-null-character[p1374] parse error[p1370]. Append a U+FFFD REPLACEMENT CHARACTER character to the current DOCTYPE token's name.

↪ **EOF**

This is an eof-in-doctype[p1371] parse error[p1370]. Set the current DOCTYPE token's *force-quirks flag*[p1387] to *on*. Emit the current DOCTYPE token. Emit an end-of-file token.

↪ **Anything else**

Append the current input character[p1382] to the current DOCTYPE token's name.

13.2.5.56 After DOCTYPE name state §[p1406]

Consume the next input character[p1382]:

↪ **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[p1388]. Emit the current DOCTYPE token.

↪ **EOF**

This is an eof-in-doctype[p1371] parse error[p1370]. Set the current DOCTYPE token's *force-quirks flag*[p1387] to *on*. Emit the current DOCTYPE token. Emit an end-of-file token.

↪ **Anything else**

If the six characters starting from the current input character[p1382] are an ASCII case-insensitive match for "PUBLIC", then consume those characters and switch to the after DOCTYPE public keyword state[p1406].

Otherwise, if the six characters starting from the current input character[p1382] are an ASCII case-insensitive match for "SYSTEM", then consume those characters and switch to the after DOCTYPE system keyword state[p1409].

Otherwise, this is an invalid-character-sequence-after-doctype-name[p1371] parse error[p1370]. Set the current DOCTYPE token's *force-quirks flag*[p1387] to *on*. Reconsume[p1387] in the bogus DOCTYPE state[p1411].

13.2.5.57 After DOCTYPE public keyword state §[p1406]

Consume the next input character[p1382]:

↪ **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[p1407].

↪ **U+0022 QUOTATION MARK (")**

This is a missing-whitespace-after-doctype-public-keyword[p1372] parse error[p1370]. Set the current DOCTYPE token's public identifier to the empty string (not missing), then switch to the DOCTYPE public identifier (double-quoted) state[p1407].

↪ **U+0027 APOSTROPHE (')**

This is a missing-whitespace-after-doctype-public-keyword[p1372] parse error[p1370]. Set the current DOCTYPE token's public identifier to the empty string (not missing), then switch to the DOCTYPE public identifier (single-quoted) state[p1407].

↪ **U+003E GREATER-THAN SIGN (>)**

This is a missing-doctype-public-identifier[p1372] parse error[p1370]. Set the current DOCTYPE token's *force-quirks flag*[p1387] to *on*. Switch to the data state[p1388]. Emit the current DOCTYPE token.

↪ **EOF**

This is an eof-in-doctype[p1371] parse error[p1370]. Set the current DOCTYPE token's *force-quirks flag*[p1387] to *on*. Emit the current DOCTYPE token. Emit an end-of-file token.

↪ **Anything else**

This is a missing-quote-before-doctype-public-identifier[p1372] parse error[p1370]. Set the current DOCTYPE token's *force-quirks flag*[p1387] to *on*. Reconsume[p1387] in the bogus DOCTYPE state[p1411].

13.2.5.58 Before DOCTYPE public identifier state §[p1407]

Consume the next input character[p1382]:

↪ **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 current DOCTYPE token's public identifier to the empty string (not missing), then switch to the DOCTYPE public identifier (double-quoted) state[p1407].

↪ **U+0027 APOSTROPHE (')**

Set the current DOCTYPE token's public identifier to the empty string (not missing), then switch to the DOCTYPE public identifier (single-quoted) state[p1407].

↪ **U+003E GREATER-THAN SIGN (>)**

This is a missing-doctype-public-identifier[p1372] parse error[p1370]. Set the current DOCTYPE token's *force-quirks flag*[p1387] to *on*. Switch to the data state[p1388]. Emit the current DOCTYPE token.

↪ **EOF**

This is an eof-in-doctype[p1371] parse error[p1370]. Set the current DOCTYPE token's *force-quirks flag*[p1387] to *on*. Emit the current DOCTYPE token. Emit an end-of-file token.

↪ **Anything else**

This is a missing-quote-before-doctype-public-identifier[p1372] parse error[p1370]. Set the current DOCTYPE token's *force-quirks flag*[p1387] to *on*. Reconsume[p1387] in the bogus DOCTYPE state[p1411].

13.2.5.59 DOCTYPE public identifier (double-quoted) state §[p1407]

Consume the next input character[p1382]:

↪ **U+0022 QUOTATION MARK (")**

Switch to the after DOCTYPE public identifier state[p1408].

↪ **U+0000 NULL**

This is an unexpected-null-character[p1374] parse error[p1370]. 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[p1370] parse error[p1370]. Set the current DOCTYPE token's *force-quirks flag*[p1387] to *on*. Switch to the data state[p1388]. Emit the current DOCTYPE token.

↪ **EOF**

This is an eof-in-doctype[p1371] parse error[p1370]. Set the current DOCTYPE token's *force-quirks flag*[p1387] to *on*. Emit the current DOCTYPE token. Emit an end-of-file token.

↪ **Anything else**

Append the current input character[p1382] to the current DOCTYPE token's public identifier.

13.2.5.60 DOCTYPE public identifier (single-quoted) state §[p1407]

Consume the next input character[p1382]:

↪ **U+0027 APOSTROPHE (')**

Switch to the after DOCTYPE public identifier state[p1408].

↪ **U+0000 NULL**

This is an unexpected-null-character[p1374] parse error[p1370]. 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[p1370] parse error[p1370]. Set the current DOCTYPE token's *force-quirks flag*[p1387] to *on*. Switch to the data state[p1388]. Emit the current DOCTYPE token.

↪ **EOF**

This is an eof-in-doctype[p1371] parse error[p1370]. Set the current DOCTYPE token's *force-quirks flag*[p1387] to *on*. Emit the current DOCTYPE token. Emit an end-of-file token.

↪ **Anything else**

Append the current input character[p1382] to the current DOCTYPE token's public identifier.

13.2.5.61 After DOCTYPE public identifier state §[p1408]

Consume the next input character[p1382]:

↪ **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[p1408].

↪ **U+003E GREATER-THAN SIGN (>)**

Switch to the data state[p1388]. Emit the current DOCTYPE token.

↪ **U+0022 QUOTATION MARK (")**

This is a missing-whitespace-between-doctype-public-and-system-identifiers[p1373] parse error[p1370]. Set the current DOCTYPE token's system identifier to the empty string (not missing), then switch to the DOCTYPE system identifier (double-quoted) state[p1410].

↪ **U+0027 APOSTROPHE (')**

This is a missing-whitespace-between-doctype-public-and-system-identifiers[p1373] parse error[p1370]. Set the current DOCTYPE token's system identifier to the empty string (not missing), then switch to the DOCTYPE system identifier (single-quoted) state[p1410].

↪ **EOF**

This is an eof-in-doctype[p1371] parse error[p1370]. Set the current DOCTYPE token's *force-quirks flag*[p1387] to *on*. Emit the current DOCTYPE token. Emit an end-of-file token.

↪ **Anything else**

This is a missing-quote-before-doctype-system-identifier[p1372] parse error[p1370]. Set the current DOCTYPE token's *force-quirks flag*[p1387] to *on*. Reconsume[p1387] in the bogus DOCTYPE state[p1411].

13.2.5.62 Between DOCTYPE public and system identifiers state §[p1408]

Consume the next input character[p1382]:

↪ **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[p1388]. Emit the current DOCTYPE token.

↪ **U+0022 QUOTATION MARK (")**

Set the current DOCTYPE token's system identifier to the empty string (not missing), then switch to the DOCTYPE system identifier (double-quoted) state[p1410].

↪ **U+0027 APOSTROPHE (')**

Set the current DOCTYPE token's system identifier to the empty string (not missing), then switch to the DOCTYPE system identifier (single-quoted) state[p1410].

↪ **EOF**

This is an eof-in-doctype[p1371] parse error[p1370]. Set the current DOCTYPE token's *force-quirks flag*[p1387] to *on*. Emit the current DOCTYPE token. Emit an end-of-file token.

↪ **Anything else**

This is a missing-quote-before-doctype-system-identifier[p1372] parse error[p1370]. Set the current DOCTYPE token's *force-quirks flag*[p1387] to *on*. Reconsume[p1387] in the bogus DOCTYPE state[p1411].

13.2.5.63 After DOCTYPE system keyword state §[p1409]

Consume the next input character[p1382]:

↪ **U+0009 CHARACTER TABULATION (tab)**
↪ **U+000A LINE FEED (LF)**
↪ **U+000C FORM FEED (FF)**
↪ **U+0020 SPACE**

Switch to the before DOCTYPE system identifier state[p1409].

↪ **U+0022 QUOTATION MARK (")**

This is a missing-whitespace-after-doctype-system-keyword[p1372] parse error[p1370]. Set the current DOCTYPE token's system identifier to the empty string (not missing), then switch to the DOCTYPE system identifier (double-quoted) state[p1410].

↪ **U+0027 APOSTROPHE (')**

This is a missing-whitespace-after-doctype-system-keyword[p1372] parse error[p1370]. Set the current DOCTYPE token's system identifier to the empty string (not missing), then switch to the DOCTYPE system identifier (single-quoted) state[p1410].

↪ **U+003E GREATER-THAN SIGN (>)**

This is a missing-doctype-system-identifier[p1372] parse error[p1370]. Set the current DOCTYPE token's *force-quirks flag*[p1387] to *on*. Switch to the data state[p1388]. Emit the current DOCTYPE token.

↪ **EOF**

This is an eof-in-doctype[p1371] parse error[p1370]. Set the current DOCTYPE token's *force-quirks flag*[p1387] to *on*. Emit the current DOCTYPE token. Emit an end-of-file token.

↪ **Anything else**

This is a missing-quote-before-doctype-system-identifier[p1372] parse error[p1370]. Set the current DOCTYPE token's *force-quirks flag*[p1387] to *on*. Reconsume[p1387] in the bogus DOCTYPE state[p1411].

13.2.5.64 Before DOCTYPE system identifier state §[p1409]

Consume the next input character[p1382]:

↪ **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 current DOCTYPE token's system identifier to the empty string (not missing), then switch to the DOCTYPE system identifier (double-quoted) state[p1410].

↪ **U+0027 APOSTROPHE (')**

Set the current DOCTYPE token's system identifier to the empty string (not missing), then switch to the DOCTYPE system identifier (single-quoted) state[p1410].

↪ **U+003E GREATER-THAN SIGN (>)**

This is a missing-doctype-system-identifier[p1372] parse error[p1370]. Set the current DOCTYPE token's *force-quirks flag*[p1387] to *on*. Switch to the data state[p1388]. Emit the current DOCTYPE token.

↪ **EOF**

This is an eof-in-doctype[p1371] parse error[p1370]. Set the current DOCTYPE token's *force-quirks flag*[p1387] to *on*. Emit the current DOCTYPE token. Emit an end-of-file token.

↪ **Anything else**

This is a missing-quote-before-doctype-system-identifier[p1372] parse error[p1370]. Set the current DOCTYPE token's *force-quirks flag*[p1387] to *on*. Reconsume[p1387] in the bogus DOCTYPE state[p1411].

13.2.5.65 DOCTYPE system identifier (double-quoted) state §[p14 10]

Consume the next input character[p1382]:

↪ **U+0022 QUOTATION MARK (")**

Switch to the after DOCTYPE system identifier state[p1411].

↪ **U+0000 NULL**

This is an unexpected-null-character[p1374] parse error[p1370]. Append a U+FFFD REPLACEMENT CHARACTER character to the current DOCTYPE token's system identifier.

↪ **U+003E GREATER-THAN SIGN (>)**

This is an abrupt-doctype-system-identifier[p1370] parse error[p1370]. Set the current DOCTYPE token's *force-quirks flag*[p1387] to *on*. Switch to the data state[p1388]. Emit the current DOCTYPE token.

↪ **EOF**

This is an eof-in-doctype[p1371] parse error[p1370]. Set the current DOCTYPE token's *force-quirks flag*[p1387] to *on*. Emit the current DOCTYPE token. Emit an end-of-file token.

↪ **Anything else**

Append the current input character[p1382] to the current DOCTYPE token's system identifier.

13.2.5.66 DOCTYPE system identifier (single-quoted) state §[p14 10]

Consume the next input character[p1382]:

↪ **U+0027 APOSTROPHE (')**

Switch to the after DOCTYPE system identifier state[p1411].

↪ **U+0000 NULL**

This is an unexpected-null-character[p1374] parse error[p1370]. Append a U+FFFD REPLACEMENT CHARACTER character to the current DOCTYPE token's system identifier.

↪ **U+003E GREATER-THAN SIGN (>)**

This is an abrupt-doctype-system-identifier[p1370] parse error[p1370]. Set the current DOCTYPE token's *force-quirks flag*[p1387] to *on*. Switch to the data state[p1388]. Emit the current DOCTYPE token.

↪ **EOF**

This is an eof-in-doctype[p1371] parse error[p1370]. Set the current DOCTYPE token's *force-quirks flag*[p1387] to *on*. Emit the current DOCTYPE token. Emit an end-of-file token.

↪ **Anything else**

Append the current input character[p1382] to the current DOCTYPE token's system identifier.

13.2.5.67 After DOCTYPE system identifier state §p14 11

Consume the next input character[p1382]:

↪ **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[p1388]. Emit the current DOCTYPE token.

↪ **EOF**
This is an eof-in-doctype[p1371] parse error[p1370]. Set the current DOCTYPE token's *force-quirks flag*[p1387] to *on*. Emit the current DOCTYPE token. Emit an end-of-file token.

↪ **Anything else**
This is an unexpected-character-after-doctype-system-identifier[p1373] parse error[p1370]. Reconsume[p1387] in the bogus DOCTYPE state[p1411]. (This does *not* set the current DOCTYPE token's *force-quirks flag*[p1387] to *on*.)

13.2.5.68 Bogus DOCTYPE state §p14 11

Consume the next input character[p1382]:

↪ **U+003E GREATER-THAN SIGN (>)**
Switch to the data state[p1388]. Emit the current DOCTYPE token.

↪ **U+0000 NULL**
This is an unexpected-null-character[p1374] parse error[p1370]. Ignore the character.

↪ **EOF**
Emit the current DOCTYPE token. Emit an end-of-file token.

↪ **Anything else**
Ignore the character.

13.2.5.69 CDATA section state §p14 11

Consume the next input character[p1382]:

↪ **U+005D RIGHT SQUARE BRACKET (])**
Switch to the CDATA section bracket state[p1411].

↪ **EOF**
This is an eof-in-cdata[p1371] parse error[p1370]. Emit an end-of-file token.

↪ **Anything else**
Emit the current input character[p1382] as a character token.

Note

U+0000 NULL characters are handled in the tree construction stage, as part of the rules for parsing tokens in foreign content[p1455], which is the only place where CDATA sections can appear.

13.2.5.70 CDATA section bracket state §p14 11

Consume the next input character[p1382]:

↪ **U+005D RIGHT SQUARE BRACKET (])**

Switch to the CDATA section end state[p1412].

↪ **Anything else**

Emit a U+005D RIGHT SQUARE BRACKET character token. Reconsume[p1387] in the CDATA section state[p1411].

13.2.5.71 CDATA section end state §[p14] 12

Consume the next input character[p1382]:

↪ **U+005D RIGHT SQUARE BRACKET (])**

Emit a U+005D RIGHT SQUARE BRACKET character token.

↪ **U+003E GREATER-THAN SIGN (>)**

Switch to the data state[p1388].

↪ **Anything else**

Emit two U+005D RIGHT SQUARE BRACKET character tokens. Reconsume[p1387] in the CDATA section state[p1411].

13.2.5.72 Processing instruction open state §[p14] 12

Consume the next input character[p1382]:

↪ **ASCII alpha**

↪ **U+005F LOW LINE (_)**

Reconsume[p1387] in the processing instruction target state[p1412].

↪ **EOF**

This is an eof-in-processing-instruction[p1371] parse error[p1370]. Emit an end-of-file token.

↪ **Anything else**

This is an invalid-first-character-of-processing-instruction-target[p1371] parse error[p1370]. Convert the temporary buffer to a comment[p1388]. Reconsume[p1387] in the bogus comment state[p1401].

13.2.5.73 Processing instruction target state §[p14] 12

Consume the next input character[p1382]:

↪ **U+0009 CHARACTER TABULATION (tab)**

↪ **U+000A LINE FEED (LF)**

↪ **U+000C FORM FEED (FF)**

↪ **U+0020 SPACE**

↪ **U+003F QUESTION MARK (?)**

↪ **U+003E GREATER-THAN SIGN (>)**

Let *target* be the concatenation of the code points in the *temporary buffer*[p1387], in the order they were added to the buffer.

If *target* is an ASCII case-insensitive match for "`xml`" or "`xml-stylesheet`":

1. This is a disallowed-processing-instruction-target[p1371] parse error[p1370].
2. Convert the temporary buffer to a comment[p1388].
3. Reconsume[p1387] in the bogus comment state[p1401].

Otherwise:

1. Create a processing instruction token whose target is *target* and data is the empty string.
2. Reconsume[p1387] in the after processing instruction target state[p1413].

↪ **ASCII alphanumeric**

↪ **U+002D HYPHEN-MINUS (-)**

↪ **U+005F LOW LINE (_)**

Append the current input character[p1382] to the *temporary buffer*[p1387].

↪ **EOF**

This is an eof-in-processing-instruction[p1371] parse error[p1370]. Emit an end-of-file token.

↪ **Anything else**

This is an invalid-processing-instruction-target[p1372] parse error[p1370]. Convert the temporary buffer to a comment[p1388]. Reconsume[p1387] in the bogus comment state[p1401].

13.2.5.74 After processing instruction target state §[p14 13]

Consume the next input character[p1382]:

↪ **U+0009 CHARACTER TABULATION (tab)**

↪ **U+000A LINE FEED (LF)**

↪ **U+000C FORM FEED (FF)**

↪ **U+0020 SPACE**

Ignore the character.

↪ **Anything else**

Reconsume[p1387] in the processing instruction data state[p1413].

13.2.5.75 Processing instruction data state §[p14 13]

Consume the next input character[p1382]:

↪ **U+003F QUESTION MARK (?)**

Switch to the processing instruction questionable state[p1413].

↪ **U+003E GREATER-THAN SIGN (>)**

Switch to the data state[p1388]. Emit the current processing instruction token.

↪ **EOF**

This is an eof-in-processing-instruction[p1371] parse error[p1370]. Emit an end-of-file token.

↪ **Anything else**

Append the current input character[p1382] to the current processing instruction token's data.

13.2.5.76 Processing instruction questionable state §[p14 13]

Consume the next input character[p1382]:

↪ **U+003E GREATER-THAN SIGN (>)**

Switch to the data state[p1388]. Emit the current processing instruction token.

↪ **EOF**

This is an eof-in-processing-instruction[p1371] parse error[p1370]. Emit an end-of-file token.

↪ **Anything else**

Append U+003F (?) to the current processing instruction token's data. Reconsume[p1387] in the processing instruction data state[p1413].

13.2.5.77 Character reference state §[p14]

Set the *temporary buffer*[p1387] to the empty string. Append a U+0026 AMPERSAND (&) character to the *temporary buffer*[p1387]. Consume the next input character[p1382]:

↪ **ASCII alphanumeric**

Reconsume[p1387] in the named character reference state[p1414].

↪ **U+0023 NUMBER SIGN (#)**

Append the current input character[p1382] to the *temporary buffer*[p1387]. Switch to the numeric character reference state[p1415].

↪ **Anything else**

Flush code points consumed as a character reference[p1388]. Reconsume[p1387] in the *return state*[p1387].

13.2.5.78 Named character reference state §[p14]

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[p1474] table. Append each character to the *temporary buffer*[p1387] when it's consumed.

↪ **If there is a match**

If the character reference was consumed as part of an attribute[p1388], and the last character matched is not a U+003B SEMICOLON character (;), and the next input character[p1382] is either a U+003D EQUALS SIGN character (=) or an ASCII alphanumeric, then, for historical reasons, flush code points consumed as a character reference[p1388] and switch to the *return state*[p1387].

Otherwise:

1. If the last character matched is not a U+003B SEMICOLON character (;), then this is a missing-semicolon-after-character-reference[p1372] parse error[p1370].
2. Set the *temporary buffer*[p1387] 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[p1474] table) to the *temporary buffer*[p1387].
3. Flush code points consumed as a character reference[p1388]. Switch to the *return state*[p1387].

↪ **Otherwise**

Flush code points consumed as a character reference[p1388]. Switch to the ambiguous ampersand state[p1414].

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 ∉ 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).

13.2.5.79 Ambiguous ampersand state §[p14]

Consume the next input character[p1382]:

↪ **ASCII alphanumeric**

If the character reference was consumed as part of an attribute[p1388], then append the current input character[p1382] to the current attribute's value. Otherwise, emit the current input character[p1382] as a character token.

↪ **U+003B SEMICOLON (;)**

This is an unknown-named-character-reference[p1374] parse error[p1370]. Reconsume[p1387] in the *return state*[p1387].

↪ **Anything else**

Reconsume[p1387] in the *return state*[p1387].

13.2.5.80 Numeric character reference state §[p14 15]

Set the ***character reference code*** to zero (0).

Consume the next input character[p1382]:

↪ **U+0078 LATIN SMALL LETTER X**

↪ **U+0058 LATIN CAPITAL LETTER X**

Append the current input character[p1382] to the *temporary buffer*[p1387]. Switch to the hexadecimal character reference start state[p1415].

↪ **ASCII digit**

Reconsume[p1387] in the decimal character reference state[p1415].

↪ **Anything else**

This is an absence-of-digits-in-numeric-character-reference[p1370] parse error[p1370]. Flush code points consumed as a character reference[p1388]. Reconsume[p1387] in the *return state*[p1387].

13.2.5.81 Hexadecimal character reference start state §[p14 15]

Consume the next input character[p1382]:

↪ **ASCII hex digit**

Reconsume[p1387] in the hexadecimal character reference state[p1415].

↪ **Anything else**

This is an absence-of-digits-in-numeric-character-reference[p1370] parse error[p1370]. Flush code points consumed as a character reference[p1388]. Reconsume[p1387] in the *return state*[p1387].

13.2.5.82 Hexadecimal character reference state §[p14 15]

Consume the next input character[p1382]:

↪ **ASCII digit**

Multiply the *character reference code*[p1415] by 16. Add a numeric version of the current input character[p1382] (subtract 0x0030 from the character's code point) to the *character reference code*[p1415].

↪ **ASCII upper hex digit**

Multiply the *character reference code*[p1415] by 16. Add a numeric version of the current input character[p1382] as a hexadecimal digit (subtract 0x0037 from the character's code point) to the *character reference code*[p1415].

↪ **ASCII lower hex digit**

Multiply the *character reference code*[p1415] by 16. Add a numeric version of the current input character[p1382] as a hexadecimal digit (subtract 0x0057 from the character's code point) to the *character reference code*[p1415].

↪ **U+003B SEMICOLON (;)**

Switch to the numeric character reference end state[p1416].

↪ **Anything else**

This is a missing-semicolon-after-character-reference[p1372] parse error[p1370]. Reconsume[p1387] in the numeric character reference end state[p1416].

13.2.5.83 Decimal character reference state §[p14 15]

Consume the next input character[p1382]:

↪ **ASCII digit**

Multiply the *character reference code*[p1415] by 10. Add a numeric version of the current input character[p1382] (subtract 0x0030 from the character's code point) to the *character reference code*[p1415].

↪ **U+003B SEMICOLON (;)**

Switch to the numeric character reference end state[p1416].

↪ **Anything else**

This is a missing-semicolon-after-character-reference[p1372] parse error[p1370]. Reconsume[p1387] in the numeric character reference end state[p1416].

13.2.5.84 Numeric character reference end state §[p1416]

Check the *character reference code*[p1415]:

- If the number is 0x00, then this is a null-character-reference[p1373] parse error[p1370]. Set the *character reference code*[p1415] to 0xFFFD.
- If the number is greater than 0x10FFFF, then this is a character-reference-outside-unicode-range[p1370] parse error[p1370]. Set the *character reference code*[p1415] to 0xFFFD.
- If the number is a surrogate, then this is a surrogate-character-reference[p1373] parse error[p1370]. Set the *character reference code*[p1415] to 0xFFFD.
- If the number is a noncharacter, then this is a noncharacter-character-reference[p1373] parse error[p1370].
- If the number is 0x0D, or a control that's not ASCII whitespace, then this is a control-character-reference[p1370] parse error[p1370]. 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*[p1415] to the number in the second column of that row.

Number	Code point	
0x80	0x20AC	EURO SIGN (€)
0x82	0x201A	SINGLE LOW-9 QUOTATION MARK (‚)
0x83	0x0192	LATIN SMALL LETTER F WITH HOOK (ƒ)
0x84	0x201E	DOUBLE LOW-9 QUOTATION MARK („)
0x85	0x2026	HORIZONTAL ELLIPSIS (…)
0x86	0x2020	DAGGER (†)
0x87	0x2021	DOUBLE DAGGER (‡)
0x88	0x02C6	MODIFIER LETTER CIRCUMFLEX ACCENT (ˆ)
0x89	0x2030	PER MILLE SIGN (‰)
0x8A	0x0160	LATIN CAPITAL LETTER S WITH CARON (Š)
0x8B	0x2039	SINGLE LEFT-POINTING ANGLE QUOTATION MARK (‹)
0x8C	0x0152	LATIN CAPITAL LIGATURE OE (Œ)
0x8E	0x017D	LATIN CAPITAL LETTER Z WITH CARON (Ž)
0x91	0x2018	LEFT SINGLE QUOTATION MARK (‘)
0x92	0x2019	RIGHT SINGLE QUOTATION MARK (’)
0x93	0x201C	LEFT DOUBLE QUOTATION MARK (“)
0x94	0x201D	RIGHT DOUBLE QUOTATION MARK (”)
0x95	0x2022	BULLET (•)
0x96	0x2013	EN DASH (–)
0x97	0x2014	EM DASH (—)
0x98	0x02DC	SMALL TILDE (˜)
0x99	0x2122	TRADE MARK SIGN (™)
0x9A	0x0161	LATIN SMALL LETTER S WITH CARON (š)
0x9B	0x203A	SINGLE RIGHT-POINTING ANGLE QUOTATION MARK (›)
0x9C	0x0153	LATIN SMALL LIGATURE OE (œ)
0x9E	0x017E	LATIN SMALL LETTER Z WITH CARON (ž)
0x9F	0x0178	LATIN CAPITAL LETTER Y WITH DIAERESIS (Ÿ)

Set the *temporary buffer*[p1387] to the empty string. Append a code point equal to the *character reference code*[p1415] to the *temporary buffer*[p1387]. Flush code points consumed as a character reference[p1388]. Switch to the *return state*[p1387].

13.2.6 Tree construction §[p1417]

The input to the tree construction stage is a sequence of tokens from the tokenization[p1387] stage. The tree construction stage is associated with a DOM `Document`[p135] 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`[p135] 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[p1383] is empty**

↪ **If the adjusted current node[p1384] is an element in the HTML namespace**

↪ **If the adjusted current node[p1384] is a MathML text integration point[p1417] and the token is a start tag whose tag name is neither "mglyph" nor "malignmark"**

↪ **If the adjusted current node[p1384] is a MathML text integration point[p1417] and the token is a character token**

↪ **If the adjusted current node[p1384] is a MathML `annotation-xml` element and the token is a start tag whose tag name is "svg"**

↪ **If the adjusted current node[p1384] is an HTML integration point[p1417] and the token is a start tag**

↪ **If the adjusted current node[p1384] is an HTML integration point[p1417] 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[p1382] in HTML content.

↪ **Otherwise**

Process the token according to the rules given in the section for parsing tokens in foreign content[p1455].

The **next token** is the token that is about to be processed by the tree construction dispatcher[p1417] (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 "`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 "`application/xhtml+xml`"
- An SVG `foreignObject` element
- An SVG `desc` element
- An SVG `title` element

Note

If the node in question is the parser's fragment context element[p1386], then the start tag token for that element is the "fake" token created by the HTML fragment parsing algorithm[p1473].

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 §p14 18

An **insertion location** is a tuple consisting of a **target parent** (a Node) and a **reference child** (a Node or null).

While the parser is processing a token, it can enable or disable **foster parenting**. This affects the following algorithm.

To find the **appropriate place for inserting a node**, given an optional Node *overrideTarget* (default null), perform the following steps. They return an insertion location[p1418].

1. Let *targetParent* be *overrideTarget* if it is not null; otherwise the current node[p1384].
2. Let *referenceChild* be null.
3. If foster parenting[p1418] is enabled and *targetParent* is a `table`[p496], `tbody`[p507], `tfoot`[p509], `thead`[p508], or `tr`[p510] element:

   Note

   Foster parenting happens when content is misnested in tables.

   1. Let *lastTemplateOrTable* be the last `template`[p705] or `table`[p496] element in the stack of open elements[p1383] if there is one; otherwise null.
   2. If *lastTemplateOrTable* is null, then return (the first element in the stack of open elements[p1383] (the `html`[p179] element), null). (fragment case[p1473])
   3. If *lastTemplateOrTable* is a `template`[p705] element, then set *targetParent* to *lastTemplateOrTable*.
   4. Otherwise, if *lastTemplateOrTable*'s parent is not null, then set *targetParent* to *lastTemplateOrTable*'s parent and set *referenceChild* to *lastTemplateOrTable*.
   5. Otherwise, set *targetParent* to the element immediately above *lastTemplateOrTable* in the stack of open elements[p1383].

   Note

   These steps are involved in part because it's possible for elements, the `table`[p496] element in this case in particular, to have been moved by a script around in the DOM, or indeed removed from the DOM entirely, after the element was inserted by the parser.

4. If *targetParent* is not a `template`[p705] element, then return (*targetParent*, *referenceChild*).
5. Let *patchInsertionTarget* be *targetParent*'s insertion target[p707].
6. If *patchInsertionTarget* is null, then return (*targetParent*'s template contents[p707], null).
7. If *targetParent*'s insertion end marker[p707] is not null and its parent is *patchInsertionTarget*, then return (*patchInsertionTarget*, *targetParent*'s insertion end marker[p707]).

   Note

   If the end marker has been removed or moved to another node, nodes are instead appended to patchInsertionTarget *in the following step.*

8. Return (*patchInsertionTarget*, null).

To **create an element for a token**, given a token *token*, a string *namespace*, a Node object *intendedParent*, and an optional string-or-null *prefix* (default null):

1. If the active speculative HTML parser[p1459] is not null, then return the result of creating a speculative mock element[p1461] given *namespace*, *token*'s tag name, and *token*'s attributes.
2. Otherwise, optionally create a speculative mock element[p1461] given *namespace*, *token*'s tag name, and *token*'s attributes.

   Note

   The result is not used. This step allows for a speculative fetch[p1460] to be initiated from non-speculative parsing. The fetch is still speculative at this point, because, for example, by the time the element is inserted, intendedParent *might have been removed from the document.*

3. Let *document* be *intendedParent*'s node document.
4. Let *localName* be *token*'s tag name.
5. Let *is* be the value of the "is" attribute in *token*, if such an attribute exists; otherwise null.
6. Let *registry* be the result of looking up a custom element registry given *intendedParent*.
7. Let *definition* be the result of looking up a custom element definition given *registry*, *namespace*, *localName*, and *is*.
8. Let *willExecuteScript* be true if *definition* is non-null and the parser's fragment context element is null; otherwise false.
9. If *willExecuteScript* is true:
    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.
10. Let *element* be the result of creating an element given *document*, *localName*, *namespace*, *prefix*, *is*, *willExecuteScript*, and *registry*.

    Note

    This will cause custom element constructors to run, if willExecuteScript *is true. However, since we incremented the throw-on-dynamic-markup-insertion counter, this cannot cause new characters to be inserted into the tokenizer, or the document to be blown away.*

11. Append each attribute in the given token to *element*.

    Note

    This can enqueue a custom element callback reaction for the `attributeChangedCallback`*, which might run immediately (in the next step).*

    Note

    Even though the `is` *attribute governs the creation of a customized built-in element, it is not present during the execution of the relevant custom element constructor; it is appended in this step, along with all other attributes.*

12. If *willExecuteScript* is true:
    1. Let *queue* be the result of popping from *document*'s relevant agent's custom element reactions stack. (This will be the same element queue as was pushed above.)
    2. Invoke custom element reactions in *queue*.
    3. Decrement *document*'s throw-on-dynamic-markup-insertion counter.
13. 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. Similarly, if *element* has an `xmlns:xlink` attribute in the XMLNS namespace whose value is not the XLink Namespace, that is a parse error.
14. If *element* is a resettable element and not a form-associated custom element, then invoke its reset algorithm. (This initializes the element's value and checkedness based on the element's attributes.)
15. If *element* is a form-associated element and not a form-associated custom element, the `form` element pointer is not null, the parser is not parsing template contents, the parser's fragment context element is null, *element* is either not listed or doesn't have a `form` attribute, and the *intendedParent* is in the same tree as the element pointed to by the `form` element pointer, then associate *element* with the `form` element pointed to by the `form` element pointer and set *element*'s parser inserted flag.
16. Return *element*.

To compute the **adjusted insertion location** with an optional insertion location *insertionLocation* (default null):

1. Let *overrideTarget* be null if *insertionLocation* is null; otherwise *insertionLocation*'s target parent.
2. Let *adjustedInsertionLocation* be the appropriate place for inserting a node given *overrideTarget*.

3. If *adjustedInsertionLocation*'s target parent[p1418] is the first element in the stack of open elements[p1383] and the parser's root insertion target[p1386] is non-null, then set *adjustedInsertionLocation* to (the parser's root insertion target[p1386], null).
4. Return *adjustedInsertionLocation*.

To **insert an element at the adjusted insertion location** with an element *element* and an optional insertion location[p1418] *insertionLocation* (default null):

1. Let (*targetParent*, *referenceChild*) be the adjusted insertion location[p1419] given *insertionLocation*.
2. If any of the following are true:
   - *element*'s parent is non-null;
   - *element* is a host-including inclusive ancestor of *targetParent*; or
   - *targetParent* is a `Document`[p135] node that already has an element child,

   then return.
3. Assert: ensure pre-insert validity given *element*, *targetParent*, *referenceChild*, and « » does not throw.
4. If the parser's fragment context element[p1386] is null, then push a new element queue[p806] onto *element*'s relevant agent[p1141]'s custom element reactions stack[p806].
5. Insert *element* into *targetParent* before *referenceChild*.
6. If the parser's fragment context element[p1386] is null, then pop the element queue[p806] from *element*'s relevant agent[p1141]'s custom element reactions stack[p806], and invoke custom element reactions[p807] in that queue.

To **insert a foreign element**, given a token *token*, a string *namespace*, and a boolean *onlyAddToElementStack*:

1. Let *adjustedInsertionLocation* be the appropriate place for inserting a node[p1418].
2. Let *element* be the result of creating an element for the token[p1418] given *token*, *namespace*, and *adjustedInsertionLocation*'s target parent[p1418].
3. If *onlyAddToElementStack* is false, then run insert an element at the adjusted insertion location[p1420] with *element*.
4. Push *element* onto the stack of open elements[p1383] so that it is the new current node[p1384].
5. Return *element*.

To **insert an HTML element** given a token *token*: insert a foreign element[p1420] given *token*, the HTML namespace, and false.

When the steps below require the user agent to **adjust MathML attributes** for a token, then, if the token has an attribute named `definitionurl`, change its name to `definitionURL` (note the case difference).

When the steps below require the user agent to **adjust SVG attributes** for a token, then, for each attribute on the token whose attribute name is one of the ones in the first column of the following table, change the attribute's name to the name given in the corresponding cell in the second column. (This fixes the case of SVG attributes that are not all lowercase.)

Attribute name on token	Attribute name on element
`attributename`	`attributeName`
`attributetype`	`attributeType`
`basefrequency`	`baseFrequency`
`baseprofile`	`baseProfile`
`calcmode`	`calcMode`
`clippathunits`	`clipPathUnits`
`diffuseconstant`	`diffuseConstant`
`edgemode`	`edgeMode`
`filterunits`	`filterUnits`
`glyphref`	`glyphRef`
`gradienttransform`	`gradientTransform`

Attribute name on token	Attribute name on element
gradientunits	gradientUnits
kernelmatrix	kernelMatrix
kernelunitlength	kernelUnitLength
keypoints	keyPoints
keysplines	keySplines
keytimes	keyTimes
lengthadjust	lengthAdjust
limitingconeangle	limitingConeAngle
markerheight	markerHeight
markerunits	markerUnits
markerwidth	markerWidth
maskcontentunits	maskContentUnits
maskunits	maskUnits
numoctaves	numOctaves
pathlength	pathLength
patterncontentunits	patternContentUnits
patterntransform	patternTransform
patternunits	patternUnits
pointsatx	pointsAtX
pointsaty	pointsAtY
pointsatz	pointsAtZ
preservealpha	preserveAlpha
preserveaspectratio	preserveAspectRatio
primitiveunits	primitiveUnits
refx	refX
refy	refY
repeatcount	repeatCount
repeatdur	repeatDur
requiredextensions	requiredExtensions
requiredfeatures	requiredFeatures
specularconstant	specularConstant
specularexponent	specularExponent
spreadmethod	spreadMethod
startoffset	startOffset
stddeviation	stdDeviation
stitchtiles	stitchTiles
surfacescale	surfaceScale
systemlanguage	systemLanguage
tablevalues	tableValues
targetx	targetX
targety	targetY
textlength	textLength
viewbox	viewBox
viewtarget	viewTarget
xchannelselector	xChannelSelector
ychannelselector	yChannelSelector
zoomandpan	zoomAndPan

When the steps below require the user agent to **adjust foreign attributes** for a token, then, if any of the attributes on the token match the strings given in the first column of the following table, let the attribute be a namespaced attribute, with the prefix being the string given in the corresponding cell in the second column, the local name being the string given in the corresponding cell in the third column, and the namespace being the namespace given in the corresponding cell in the fourth column. (This fixes the use of namespaced attributes, in particular lang attributes in the XML namespace.)

Attribute name	Prefix	Local name	Namespace
xlink:actuate	xlink	actuate	XLink namespace

Attribute name	Prefix	Local name	Namespace
xlink:arcrole	xlink	arcrole	XLink namespace
xlink:href	xlink	href	XLink namespace
xlink:role	xlink	role	XLink namespace
xlink:show	xlink	show	XLink namespace
xlink:title	xlink	title	XLink namespace
xlink:type	xlink	type	XLink namespace
xml:lang	xml	lang	XML namespace
xml:space	xml	space	XML namespace
xmlns	(none)	xmlns	XMLNS namespace
xmlns:xlink	xmlns	xlink	XMLNS namespace

When the steps below require the user agent to **insert a character** while processing a token, the user agent must run the following steps:

1. Let *data* be the characters passed to the algorithm, or, if no characters were explicitly specified, the character of the character token being processed.
2. Let (*targetParent*, *referenceChild*) be the adjusted insertion location[p1419].
3. If *targetParent* is a Document[p135] node, then return.

   Note

   The DOM will not let Document[p135] nodes have Text node children, so they are dropped on the floor.
4. Let *previousSibling* be *referenceChild*'s previous sibling if *referenceChild* is non-null; otherwise *targetParent*'s last child.
5. If *previousSibling* is a Text node, then append *data* to *previousSibling*'s data.

   Otherwise, let *text* be the result of creating a text node given *targetParent*'s node document and *data*, and insert *text* into *targetParent* before *referenceChild*.

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.

Input	Number of Text nodes

`A<script>`
`var script = document.getElementsByTagName('script')[0];`
`document.body.removeChild(script);`
`</script>B`
	One Text node in the document, containing "AB".

`A<script>`
`var text = document.createTextNode('B');`
`document.body.appendChild(text);`
`</script>C`
	Three Text nodes; "A" before the script, the script's contents, and "BC" after the script (the parser appends to the Text node created by the script).

`A<script>`
`var text = document.getElementsByTagName('script')[0].firstChild;`
`text.data = 'B';`
`document.body.appendChild(text);`
`</script>C`
	Two adjacent Text nodes in the document, containing "A" and "BC".

`A<table>B<tr>C</tr>D</table>`	One Text node before the table, containing "ABCD". (This is caused by foster parenting[p1418].)
`A<table><tr> B</tr> C</table>`	One Text node before the table, containing "A B C" (A-space-B-space-C). (This is caused by foster parenting[p1418].)
`A<table><tr> B</tr> </em>C</table>`	One Text node before the table, containing "A BC" (A-space-B-C), and one Text node inside the table (as a child of a tbody[p507]) with a single space character. (Space characters separated from non-space characters by non-character tokens are not affected by foster parenting[p1418], even if those other tokens then get

Input	Number of Text nodes
	ignored.)

When the steps below require the user agent to **insert a comment** while processing a comment token, with an optional insertion location[p1418] *insertionLocation* (default null), the user agent must run the following steps:

1. Let *data* be the data given in the comment token being processed.
2. Let (*targetParent*, *referenceChild*) be the adjusted insertion location[p1419] given *insertionLocation*.
3. Let *comment* be the result of creating a comment node given *targetParent*'s node document and *data*.
4. Insert *comment* into *targetParent* before *referenceChild*.

When the steps below require the user agent to **insert a processing instruction** while processing a processing instruction token, with an optional insertion location[p1418] *insertionLocation* (default null), the user agent must run the following steps:

1. Let *piTarget* be the target given in the processing instruction token being processed.
2. Let *data* be the data given in the processing instruction token being processed.
3. Let (*targetParent*, *referenceChild*) be the adjusted insertion location[p1419] given *insertionLocation*.
4. Let *pi* be the result of creating a processing instruction node given *targetParent*'s node document, *piTarget*, and *data*.
5. Insert *pi* into *targetParent* before *referenceChild*.

13.2.6.2 Parsing elements that contain only text §[p14 23]

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[p1420] for the token.
2. If the algorithm that was invoked is the generic raw text element parsing algorithm[p1423], switch the tokenizer to the RAWTEXT state[p1389]; otherwise the algorithm invoked was the generic RCDATA element parsing algorithm[p1423], switch the tokenizer to the RCDATA state[p1388].
3. Set the original insertion mode[p1382] to the current insertion mode[p1382].
4. Then, switch the insertion mode[p1382] to "text[p1443]".

13.2.6.3 Closing elements that have implied end tags §[p14 23]

When the steps below require the UA to **generate implied end tags**, then, while the current node[p1384] is a dd[p258] element, a dt[p257] element, an li[p251] element, an optgroup[p600] element, an option[p601] element, a p[p238] element, an rb[p1531] element, an rp[p288] element, an rt[p287] element, or an rtc[p1531] element, the UA must pop the current node[p1384] off the stack of open elements[p1383].

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[p1384] is a caption[p504] element, a colgroup[p505] element, a dd[p258] element, a dt[p257] element, an li[p251] element, an optgroup[p600] element, an option[p601] element, a p[p238] element, an rb[p1531] element, an rp[p288] element, an rt[p287] element, an rtc[p1531] element, a tbody[p507] element, a td[p512] element, a tfoot[p509] element, a th[p513] element, a thead[p508] element, or a tr[p510] element, the UA must pop the current node[p1384] off the stack of open elements[p1383].

13.2.6.4 The rules for parsing tokens in HTML content §[p14 24]

13.2.6.4.1 The "initial" insertion mode §[p14 24]

A Document[p135] object has an associated **parser cannot change the mode flag** (a boolean). It is initially false.

When the user agent is to apply the rules for the "initial[p1424]" insertion mode[p1382], 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[p1423] given (the Document[p135] object, null).

↪ **A processing instruction token**

Insert a processing instruction[p1423] given (the Document[p135] object, null).

↪ **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-compat[p100]", then there is a parse error[p1370].

Let *doctype* be the result of creating a doctype given the Document[p135] node, the name given in the DOCTYPE token (or the empty string if the name was missing), the public identifier given in the DOCTYPE token (or the empty string if the public identifier was missing), and the system identifier given in the DOCTYPE token (or the empty string if the system identifier was missing). Then, if the Document[p135] node has neither a DocumentType child nor an element child, append *doctype* to the Document[p135] node.

Note

This also ensures that the DocumentType node is returned as the value of the doctype attribute of the Document[p135] object.

Then, if the document is *not* an iframe srcdoc document[p405], and the parser cannot change the mode flag[p1424] is false, and the DOCTYPE token matches one of the conditions in the following list, then set the Document[p135] to quirks mode:

- The *force-quirks flag*[p1387] is set to *on*.
- The name is not "html".
- The public identifier is set to: "-//W3O//DTD W3 HTML Strict 3.0//EN//"
- The public identifier is set to: "-/W3C/DTD HTML 4.0 Transitional/EN"
- The public identifier is set to: "HTML"
- The system identifier is set to: "http://www.ibm.com/data/dtd/v11/ibmxhtml1-transitional.dtd"
- The public identifier starts with: "+//Silmaril//dtd html Pro v0r11 19970101//"
- The public identifier starts with: "-//AS//DTD HTML 3.0 asWedit + extensions//"
- The public identifier starts with: "-//AdvaSoft Ltd//DTD HTML 3.0 asWedit + extensions//"
- The public identifier starts with: "-//IETF//DTD HTML 2.0 Level 1//"
- The public identifier starts with: "-//IETF//DTD HTML 2.0 Level 2//"
- The public identifier starts with: "-//IETF//DTD HTML 2.0 Strict Level 1//"
- The public identifier starts with: "-//IETF//DTD HTML 2.0 Strict Level 2//"
- The public identifier starts with: "-//IETF//DTD HTML 2.0 Strict//"
- The public identifier starts with: "-//IETF//DTD HTML 2.0//"
- The public identifier starts with: "-//IETF//DTD HTML 2.1E//"
- The public identifier starts with: "-//IETF//DTD HTML 3.0//"
- The public identifier starts with: "-//IETF//DTD HTML 3.2 Final//"
- The public identifier starts with: "-//IETF//DTD HTML 3.2//"
- The public identifier starts with: "-//IETF//DTD HTML 3//"
- The public identifier starts with: "-//IETF//DTD HTML Level 0//"
- The public identifier starts with: "-//IETF//DTD HTML Level 1//"
- The public identifier starts with: "-//IETF//DTD HTML Level 2//"
- The public identifier starts with: "-//IETF//DTD HTML Level 3//"
- The public identifier starts with: "-//IETF//DTD HTML Strict Level 0//"
- The public identifier starts with: "-//IETF//DTD HTML Strict Level 1//"
- The public identifier starts with: "-//IETF//DTD HTML Strict Level 2//"
- The public identifier starts with: "-//IETF//DTD HTML Strict Level 3//"
- The public identifier starts with: "-//IETF//DTD HTML Strict//"
- The public identifier starts with: "-//IETF//DTD HTML//"
- The public identifier starts with: "-//Metrius//DTD Metrius Presentational//"
- The public identifier starts with: "-//Microsoft//DTD Internet Explorer 2.0 HTML Strict//"
- The public identifier starts with: "-//Microsoft//DTD Internet Explorer 2.0 HTML//"
- The public identifier starts with: "-//Microsoft//DTD Internet Explorer 2.0 Tables//"
- The public identifier starts with: "-//Microsoft//DTD Internet Explorer 3.0 HTML Strict//"
- The public identifier starts with: "-//Microsoft//DTD Internet Explorer 3.0 HTML//"
- The public identifier starts with: "-//Microsoft//DTD Internet Explorer 3.0 Tables//"
- The public identifier starts with: "-//Netscape Comm. Corp.//DTD HTML//"

- The public identifier starts with: "-//Netscape Comm. Corp.//DTD Strict HTML//"
- The public identifier starts with: "-//O'Reilly and Associates//DTD HTML 2.0//"
- The public identifier starts with: "-//O'Reilly and Associates//DTD HTML Extended 1.0//"
- The public identifier starts with: "-//O'Reilly and Associates//DTD HTML Extended Relaxed 1.0//"
- The public identifier starts with: "-//SQ//DTD HTML 2.0 HoTMetaL + extensions//"
- The public identifier starts with: "-//SoftQuad Software//DTD HoTMetaL PRO 6.0::19990601::extensions to HTML 4.0//"
- The public identifier starts with: "-//SoftQuad//DTD HoTMetaL PRO 4.0::19971010::extensions to HTML 4.0//"
- The public identifier starts with: "-//Spyglass//DTD HTML 2.0 Extended//"
- The public identifier starts with: "-//Sun Microsystems Corp.//DTD HotJava HTML//"
- The public identifier starts with: "-//Sun Microsystems Corp.//DTD HotJava Strict HTML//"
- The public identifier starts with: "-//W3C//DTD HTML 3 1995-03-24//"
- The public identifier starts with: "-//W3C//DTD HTML 3.2 Draft//"
- The public identifier starts with: "-//W3C//DTD HTML 3.2 Final//"
- The public identifier starts with: "-//W3C//DTD HTML 3.2//"
- The public identifier starts with: "-//W3C//DTD HTML 3.2S Draft//"
- The public identifier starts with: "-//W3C//DTD HTML 4.0 Frameset//"
- The public identifier starts with: "-//W3C//DTD HTML 4.0 Transitional//"
- The public identifier starts with: "-//W3C//DTD HTML Experimental 19960712//"
- The public identifier starts with: "-//W3C//DTD HTML Experimental 970421//"
- The public identifier starts with: "-//W3C//DTD W3 HTML//"
- The public identifier starts with: "-//W3O//DTD W3 HTML 3.0//"
- The public identifier starts with: "-//WebTechs//DTD Mozilla HTML 2.0//"
- The public identifier starts with: "-//WebTechs//DTD Mozilla HTML//"
- The system identifier is missing or the empty string, and the public identifier starts with: "-//W3C//DTD HTML 4.01 Frameset//"
- The system identifier is missing or the empty string, and the public identifier starts with: "-//W3C//DTD HTML 4.01 Transitional//"

Otherwise, if the document is *not* an `iframe srcdoc` document[p405], and the parser cannot change the mode flag[p1424] is false, and the DOCTYPE token matches one of the conditions in the following list, then set the Document[p135] 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 neither missing nor the empty string, and the public identifier starts with: "-//W3C//DTD HTML 4.01 Frameset//"
- The system identifier is neither missing nor the empty string, 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[p1382] to "before html[p1425]".

↪ **Anything else**

If the document is *not* an `iframe srcdoc` document[p405], then this is a parse error[p1370]; if the parser cannot change the mode flag[p1424] is false, set the Document[p135] to quirks mode.

In any case, switch the insertion mode[p1382] to "before html[p1425]", then reprocess the token.

13.2.6.4.2 The "before html" insertion mode §[p14 25]

When the user agent is to apply the rules for the "before html[p1425]" insertion mode[p1382], the user agent must handle the token as follows:

↪ **A DOCTYPE token**

Parse error[p1370]. Ignore the token.

↪ **A comment token**

Insert a comment[p1423] given (the Document[p135] object, null).

↪ **A processing instruction token**

Insert a processing instruction[p1423] given (the Document[p135] object, null).

↪ **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[p1418] in the HTML namespace, with the Document[p135] as the intended parent. Insert an element at the adjusted insertion location[p1420] with that element and (the Document[p135], null). Put this element in the stack of open elements[p1383].

Switch the insertion mode[p1382] to "before head[p1426]".

↪ **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[p1370]. Ignore the token.

↪ **Anything else**

Create an html[p179] element whose node document is the Document[p135] object. Append it to the Document[p135] object. Put this element in the stack of open elements[p1383].

Switch the insertion mode[p1382] to "before head[p1426]", then reprocess the token.

The document element can end up being removed from the Document[p135] 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 §[p1426]

When the user agent is to apply the rules for the "before head[p1426]" insertion mode[p1382], 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[p1423].

↪ **A processing instruction token**

Insert a processing instruction[p1423].

↪ **A DOCTYPE token**

Parse error[p1370]. Ignore the token.

↪ **A start tag whose tag name is "html"**

Process the token using the rules for[p1382] the "in body[p1432]" insertion mode[p1382].

↪ **A start tag whose tag name is "head"**

Insert an HTML element[p1420] for the token.

Set the head element pointer[p1386] to the newly created head[p180] element.

Switch the insertion mode[p1382] to "in head[p1427]".

↪ **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[p1370]. Ignore the token.

↪ **Anything else**

Insert an HTML element[p1420] for a "head" start tag token with no attributes.

Set the head element pointer[p1386] to the newly created head[p180] element.

Switch the insertion mode[p1382] to "in head[p1427]".

Reprocess the current token.

13.2.6.4.4 The "in head" insertion mode §[p14 27]

When the user agent is to apply the rules for the "in head[p1427]" insertion mode[p1382], 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[p1422].

↪ **A comment token**

Insert a comment[p1423].

↪ **A processing instruction token**

Insert a processing instruction[p1423].

↪ **A DOCTYPE token**

Parse error[p1370]. Ignore the token.

↪ **A start tag whose tag name is "html"**

Process the token using the rules for[p1382] the "in body[p1432]" insertion mode[p1382].

↪ **A start tag whose tag name is one of: "base", "basefont", "bgsound", "link"**

Insert an HTML element[p1420] for the token. Immediately pop the current node[p1384] off the stack of open elements[p1383].

Acknowledge the token's *self-closing flag*[p1387], if it is set.

↪ **A start tag whose tag name is "meta"**

Insert an HTML element[p1420] for the token. Immediately pop the current node[p1384] off the stack of open elements[p1383].

Acknowledge the token's *self-closing flag*[p1387], if it is set.

If the active speculative HTML parser[p1459] is null:

1. If the element has a `charset`[p197] attribute, and getting an encoding from its value results in an encoding, and the confidence[p1375] is currently *tentative*, then change the encoding[p1381] to the resulting encoding.

2. Otherwise, if the element has an `http-equiv`[p202] attribute whose value is an ASCII case-insensitive match for "`Content-Type`", and the element has a `content`[p197] attribute, and applying the algorithm for extracting a character encoding from a `meta` element[p103] to that attribute's value returns an encoding, and the confidence[p1375] is currently *tentative*, then change the encoding[p1381] to the extracted encoding.

Note

The speculative HTML parser[p1459] doesn't speculatively apply character encoding declarations in order to reduce implementation complexity.

↪ **A start tag whose tag name is "title"**

Follow the generic RCDATA element parsing algorithm[p1423].

↪ **A start tag whose tag name is "noscript", if scripting mode[p1387] is not Disabled[p1387]**

↪ **A start tag whose tag name is one of: "noframes", "style"**

Follow the generic raw text element parsing algorithm[p1423].

↪ **A start tag whose tag name is "noscript", if scripting mode[p1387] is Disabled[p1387]**

Insert an HTML element[p1420] for the token.

Switch the insertion mode[p1382] to "in head noscript[p1431]".

↪ **A start tag whose tag name is "script"**

Run these steps:

1. Let *adjustedInsertionLocation* be the appropriate place for inserting a node[p1418].
2. Create an element for the token[p1418] in the HTML namespace, with the intended parent being the node in which the *adjustedInsertionLocation* finds itself.
3. If the scripting mode[p1387] is not Fragment[p1387], then set the element's parser document[p692] to the `Document`[p135].

   Note

   The Fragment[p1387] scripting mode[p1387] treats parser-inserted[p692] scripts as if they were not parser-inserted, allowing, for example, executing scripts when applying a fragment created by `createContextualFragment()`[p1232].

4. Set the element's force async[p692] to false.

   Note

   This ensures that, if the script is external, any `document.write()`[p1223] 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's scripting mode[p1387] is Inert[p1387], then set the `script`[p685] element's already started[p692] to true. (fragment case[p1473])
6. If the parser was invoked via the `document.write()`[p1223] or `document.writeln()`[p1224] methods, then optionally set the `script`[p685] element's already started[p692] to true. (For example, the user agent might use this clause to prevent execution of cross-origin[p939] scripts inserted via `document.write()`[p1223] under slow network conditions, or when the page has already taken a long time to load.)
7. Insert an element at the adjusted insertion location[p1420] with the newly created element and *adjustedInsertionLocation*.
8. Push the element onto the stack of open elements[p1383] so that it is the new current node[p1384].
9. Switch the tokenizer to the script data state[p1389].
10. Set the original insertion mode[p1382] to the current insertion mode[p1382].
11. Switch the insertion mode[p1382] to "text[p1443]".

↪ **An end tag whose tag name is "head"**

Pop the current node[p1384] (which will be the `head`[p180] element) off the stack of open elements[p1383].

Switch the insertion mode[p1382] to "after head[p1431]".

↪ **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"**

Run these steps:

1. Let *templateStartTag* be the start tag.
2. Insert a marker[p1385] at the end of the list of active formatting elements[p1385].
3. Set the frameset-ok flag[p1387] to "not ok".
4. Switch the insertion mode[p1382] to "in template[p1452]".
5. Push "in template[p1452]" onto the stack of template insertion modes[p1383] so that it is the new current template insertion mode[p1383].
6. Let *adjustedInsertionLocation* be the appropriate place for inserting a node[p1418].
7. If *templateStartTag*'s `shadowrootmode`[p706] is not in the None[p706] state:

   1. If the parser's allow declarative shadow roots[p1387] is false, or the adjusted current node[p1384] is the topmost element in the stack of open elements[p1383], then insert an HTML element[p1420] for the token and return.

2. Let *declarativeShadowHostElement* be the adjusted current node[p1384].
3. Let *template* be the result of insert a foreign element[p1420] for *templateStartTag*, with HTML namespace and true.
4. Let *mode* be *templateStartTag*'s `shadowrootmode`[p706] attribute's value.
5. Let *delegatesFocus* be true if *templateStartTag* has a `shadowrootdelegatesfocus`[p706] attribute; otherwise false.
6. Let *serializable* be true if *templateStartTag* has a `shadowrootserializable`[p706] attribute; otherwise false.
7. Let *slotAssignment* be "`named`".
8. If *templateStartTag*'s `shadowrootslotassignment`[p706] attribute is in the Manual[p706] state, then set *slotAssignment* to "`manual`".
9. Let *clonable* be true if *templateStartTag* has a `shadowrootclonable`[p706] attribute; otherwise false.
10. If *declarativeShadowHostElement* is a shadow host, then insert an element at the adjusted insertion location[p1420] with *template* and *adjustedInsertionLocation*.
11. Otherwise:
    1. Let *registry* be null if *templateStartTag* has a `shadowrootcustomelementregistry`[p706] attribute; otherwise *declarativeShadowHostElement*'s node document's custom element registry.
    2. Attach a shadow root with *declarativeShadowHostElement*, *mode*, *delegatesFocus*, *serializable*, *slotAssignment*, *clonable*, and *registry*.

       If an exception is thrown, then catch it and:
       1. Insert an element at the adjusted insertion location[p1420] with *template* and *adjustedInsertionLocation*.
       2. The user agent may report an error to the developer console.
       3. Return.
    3. Let *shadow* be *declarativeShadowHostElement*'s shadow root.
    4. Set *shadow*'s available to element internals to true.
    5. Set *shadow*'s declarative to true.
    6. Set *template*'s template contents[p707] to *shadow*.
    7. If *templateStartTag* has a `shadowrootcustomelementregistry`[p706] attribute, then set *shadow*'s keep custom element registry null to true.

8. Otherwise, if *templateStartTag*'s `for`[p706] attribute is specified:
    1. Let *scope* be the node that contains the adjusted insertion location[p1419].

       Note

       In the fragment case[p1473], scope can be the `DocumentFragment` into which nodes are being inserted.
    2. If *scope* is a `template`[p705] element, then set *scope* to *scope*'s template contents[p707].

       Note

       This is to support patching inside plain `template`[p705] elements or those using `shadowrootmode`[p706]. Nested patching is not supported, since in this case the `template`[p705] element's template contents[p707] will have no children and prepare content patching[p708] will fail.
    3. Otherwise, if *scope* is the body element[p144] of its node document, then set *scope* to its parent.

       Note

       This is to support patching `head`[p180] with a `template`[p705] inside `body`[p212].

4. Let *template* be the result of insert a foreign element[p1420] for *templateStartTag*, with HTML namespace and true.
5. Let *success* be the result of prepare content patching[p708] given *scope* and *template*.
6. If *success* is false:
    1. Assert: *template* is the current node[p1384].
    2. Pop the current node[p1384] off the stack of open elements[p1383].
    3. Insert an element at the adjusted insertion location[p1420] with *template*.
    4. Push *template* onto the stack of open elements[p1383].

    Note

    The above steps undo the effect of the onlyAddToElementStack *argument from the insert a foreign element[p1420] call that created* template*, inserting it where it would have been inserted otherwise. This is to signal an error.*

9. Otherwise, insert an HTML element[p1420] for *templateStartTag*.

↪ **An end tag whose tag name is "template"**

If there is no `template`[p705] element on the stack of open elements[p1383], then this is a parse error[p1370]; ignore the token.

Otherwise, run these steps:

1. Generate all implied end tags thoroughly[p1423].
2. If the current node[p1384] is not a `template`[p705] element, then this is a parse error[p1370].
3. Let *template* be the last `template`[p705] element in the stack of open elements[p1383].

    Note

    This is the `template`[p705] element that will be popped from the stack below. It is not necessarily the current node[p1384], as the content of a `template`[p705] element with a `for`[p706] attribute might have left an element open.

4. If *template*'s insertion target[p707] is not null:
    1. Let *start* be *template*'s insertion start marker[p707].
    2. Let *end* be *template*'s insertion end marker[p707].
    3. Assert: *start* is not null.
    4. If *start*'s parent is not null, then remove *start*.
    5. If *end* is not null and *end*'s parent is not null, then remove *end*.
    6. Set *template*'s insertion target[p707] to null.
    7. Set *template*'s insertion start marker[p707] to null.
    8. Set *template*'s insertion end marker[p707] to null.
5. Pop elements from the stack of open elements[p1383] until a `template`[p705] element has been popped from the stack.
6. Clear the list of active formatting elements up to the last marker[p1386].
7. Pop the current template insertion mode[p1383] off the stack of template insertion modes[p1383].
8. Reset the insertion mode appropriately[p1383].

↪ **A start tag whose tag name is "head"**

↪ **Any other end tag**

Parse error[p1370]. Ignore the token.

↪ **Anything else**

Pop the current node[p1384] (which will be the `head`[p180] element) off the stack of open elements[p1383].

Switch the insertion mode[p1382] to "after head[p1431]".

Reprocess the token.

13.2.6.4.5 The "in head noscript" insertion mode §[p14 31]

When the user agent is to apply the rules for the "in head noscript[p1431]" insertion mode[p1382], the user agent must handle the token as follows:

↪ **A DOCTYPE token**

Parse error[p1370]. Ignore the token.

↪ **A start tag whose tag name is "html"**

Process the token using the rules for[p1382] the "in body[p1432]" insertion mode[p1382].

↪ **An end tag whose tag name is "noscript"**

Pop the current node[p1384] (which will be a noscript[p703] element) from the stack of open elements[p1383]; the new current node[p1384] will be a head[p180] element.

Switch the insertion mode[p1382] to "in head[p1427]".

↪ **A character token that is one of U+0009 CHARACTER TABULATION, U+000A LINE FEED (LF), U+000C FORM FEED (FF), U+000D CARRIAGE RETURN (CR), or U+0020 SPACE**

↪ **A comment token**

↪ **A processing instruction token**

↪ **A start tag whose tag name is one of: "basefont", "bgsound", "link", "meta", "noframes", "style"**

Process the token using the rules for[p1382] the "in head[p1427]" insertion mode[p1382].

↪ **An end tag whose tag name is "br"**

Act as described in the "anything else" entry below.

↪ **A start tag whose tag name is one of: "head", "noscript"**

↪ **Any other end tag**

Parse error[p1370]. Ignore the token.

↪ **Anything else**

Parse error[p1370].

Pop the current node[p1384] (which will be a noscript[p703] element) from the stack of open elements[p1383]; the new current node[p1384] will be a head[p180] element.

Switch the insertion mode[p1382] to "in head[p1427]".

Reprocess the token.

13.2.6.4.6 The "after head" insertion mode §[p14 31]

When the user agent is to apply the rules for the "after head[p1431]" insertion mode[p1382], 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[p1422].

↪ **A comment token**

Insert a comment[p1423].

↪ **A processing instruction token**

Insert a processing instruction[p1423].

↪ **A DOCTYPE token**

Parse error[p1370]. Ignore the token.

↪ **A start tag whose tag name is "html"**

Process the token using the rules for[p1382] the "in body[p1432]" insertion mode[p1382].

↪ **A start tag whose tag name is "body"**

Insert an HTML element[p1420] for the token.

Set the frameset-ok flag[p1387] to "not ok".

Switch the insertion mode[p1382] to "in body[p1432]".

↪ **A start tag whose tag name is "frameset"**

Insert an HTML element[p1420] for the token.

Switch the insertion mode[p1382] to "in frameset[p1453]".

↪ **A start tag whose tag name is one of: "base", "basefont", "bgsound", "link", "meta", "noframes", "script", "style", "template", "title"**

Parse error[p1370].

Push the node pointed to by the head element pointer[p1386] onto the stack of open elements[p1383].

Process the token using the rules for[p1382] the "in head[p1427]" insertion mode[p1382].

Remove the node pointed to by the head element pointer[p1386] from the stack of open elements[p1383]. (It might not be the current node[p1384] at this point.)

Note

The head element pointer[p1386] cannot be null at this point.

↪ **An end tag whose tag name is "template"**

Process the token using the rules for[p1382] the "in head[p1427]" insertion mode[p1382].

↪ **An end tag whose tag name is one of: "body", "html", "br"**

Act as described in the "anything else" entry below.

↪ **A start tag whose tag name is "head"**

↪ **Any other end tag**

Parse error[p1370]. Ignore the token.

↪ **Anything else**

Insert an HTML element[p1420] for a "body" start tag token with no attributes.

Set the frameset-ok flag[p1387] to "ok".

Switch the insertion mode[p1382] to "in body[p1432]".

Reprocess the current token.

13.2.6.4.7 The "in body" insertion mode §[p1432]

When the user agent is to apply the rules for the "in body[p1432]" insertion mode[p1382], the user agent must handle the token as follows:

↪ **A character token that is U+0000 NULL**

Parse error[p1370]. Ignore the token.

↪ **A character token that is one of U+0009 CHARACTER TABULATION, U+000A LINE FEED (LF), U+000C FORM FEED**

(FF), U+000D CARRIAGE RETURN (CR), or U+0020 SPACE

Reconstruct the active formatting elements[p1386], if any.

Insert the token's character[p1422].

↪ **Any other character token**

Reconstruct the active formatting elements[p1386], if any.

Insert the token's character[p1422].

Set the frameset-ok flag[p1387] to "not ok".

↪ **A comment token**

Insert a comment[p1423].

↪ **A processing instruction token**

Insert a processing instruction[p1423].

↪ **A DOCTYPE token**

Parse error[p1370]. Ignore the token.

↪ **A start tag whose tag name is "html"**

Parse error[p1370].

If there is a `template`[p705] element on the stack of open elements[p1383], 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[p1383]. 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[p1382] the "in head[p1427]" insertion mode[p1382].

↪ **A start tag whose tag name is "body"**

Parse error[p1370].

If the stack of open elements[p1383] has only one node on it, or if the second element on the stack of open elements[p1383] is not a `body`[p212] element, or if there is a `template`[p705] element on the stack of open elements[p1383], then ignore the token. (fragment case[p1473] or there is a `template`[p705] element on the stack)

Otherwise, set the frameset-ok flag[p1387] to "not ok"; then, for each attribute on the token, check to see if the attribute is already present on the `body`[p212] element (the second element) on the stack of open elements[p1383], 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[p1370].

If the stack of open elements[p1383] has only one node on it, or if the second element on the stack of open elements[p1383] is not a `body`[p212] element, then ignore the token. (fragment case[p1473] or there is a `template`[p705] element on the stack)

If the frameset-ok flag[p1387] is set to "not ok", ignore the token.

Otherwise, run the following steps:

1. Remove the second element on the stack of open elements[p1383] from its parent node, if it has one.
2. Pop all the nodes from the bottom of the stack of open elements[p1383], from the current node[p1384] up to, but not including, the root `html`[p179] element.
3. Insert an HTML element[p1420] for the token.
4. Switch the insertion mode[p1382] to "in frameset[p1453]".

↪ **An end-of-file token**

If the stack of template insertion modes[p1383] is not empty, then process the token using the rules for[p1382] the "in template[p1452]" insertion mode[p1382].

Otherwise, follow these steps:

1. If there is a node in the stack of open elements[p1383] that is not either a dd[p258] element, a dt[p257] element, an li[p251] element, an optgroup[p600] element, an option[p601] element, a p[p238] element, an rb[p1531] element, an rp[p288] element, an rt[p287] element, an rtc[p1531] element, a tbody[p507] element, a td[p512] element, a tfoot[p509] element, a th[p513] element, a thead[p508] element, a tr[p510] element, the body[p212] element, or the html[p179] element, then this is a parse error[p1370].

2. Stop parsing[p1458].

↪ **An end tag whose tag name is "body"**

If the stack of open elements[p1383] does not have a body element in scope[p1384], this is a parse error[p1370]; ignore the token.

Otherwise, if there is a node in the stack of open elements[p1383] that is not either a dd[p258] element, a dt[p257] element, an li[p251] element, an optgroup[p600] element, an option[p601] element, a p[p238] element, an rb[p1531] element, an rp[p288] element, an rt[p287] element, an rtc[p1531] element, a tbody[p507] element, a td[p512] element, a tfoot[p509] element, a th[p513] element, a thead[p508] element, a tr[p510] element, the body[p212] element, or the html[p179] element, then this is a parse error[p1370].

Switch the insertion mode[p1382] to "after body[p1453]".

↪ **An end tag whose tag name is "html"**

If the stack of open elements[p1383] does not have a body element in scope[p1384], this is a parse error[p1370]; ignore the token.

Otherwise, if there is a node in the stack of open elements[p1383] that is not either a dd[p258] element, a dt[p257] element, an li[p251] element, an optgroup[p600] element, an option[p601] element, a p[p238] element, an rb[p1531] element, an rp[p288] element, an rt[p287] element, an rtc[p1531] element, a tbody[p507] element, a td[p512] element, a tfoot[p509] element, a th[p513] element, a thead[p508] element, a tr[p510] element, the body[p212] element, or the html[p179] element, then this is a parse error[p1370].

Switch the insertion mode[p1382] to "after body[p1453]".

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", "search", "section", "summary", "ul"**

If the stack of open elements[p1383] has a p element in button scope[p1385], then close a p element[p1442].

Insert an HTML element[p1420] for the token.

↪ **A start tag whose tag name is one of: "h1", "h2", "h3", "h4", "h5", "h6"**

If the stack of open elements[p1383] has a p element in button scope[p1385], then close a p element[p1442].

If the current node[p1384] is an HTML element[p46] whose tag name is one of "h1", "h2", "h3", "h4", "h5", or "h6", then this is a parse error[p1370]; pop the current node[p1384] off the stack of open elements[p1383].

Insert an HTML element[p1420] for the token.

↪ **A start tag whose tag name is one of: "pre", "listing"**

If the stack of open elements[p1383] has a p element in button scope[p1385], then close a p element[p1442].

Insert an HTML element[p1420] for the token.

If the next token[p1417] 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[p242] blocks are ignored as an authoring convenience.)

Set the frameset-ok flag[p1387] to "not ok".

↪ **A start tag whose tag name is "form"**

If the form element pointer[p1386] is not null, and the parser is not parsing template contents[p1386], then this is a parse error[p1370]; ignore the token.

Otherwise:

1. If the stack of open elements[p1383] has a p element in button scope[p1385], then close a p element[p1442].
2. Insert an HTML element[p1420] for the token, and, if the parser is not parsing template contents[p1386], set the form element pointer[p1386] to point to the element created.

↪ **A start tag whose tag name is "li"**

Run these steps:

1. Set the frameset-ok flag[p1387] to "not ok".
2. Initialize *node* to be the current node[p1384] (the bottommost node of the stack).
3. *Loop*: If *node* is an li[p251] element, then run these substeps:
    1. Generate implied end tags[p1423], except for li[p251] elements.
    2. If the current node[p1384] is not an li[p251] element, then this is a parse error[p1370].
    3. Pop elements from the stack of open elements[p1383] until an li[p251] element has been popped from the stack.
    4. Jump to the step labeled *done* below.
4. If *node* is in the special[p1384] category, but is not an address[p230], div[p266], or p[p238] element, then jump to the step labeled *done* below.
5. Otherwise, set *node* to the previous entry in the stack of open elements[p1383] and return to the step labeled *loop*.
6. *Done*: If the stack of open elements[p1383] has a p element in button scope[p1385], then close a p element[p1442].
7. Finally, insert an HTML element[p1420] for the token.

↪ **A start tag whose tag name is one of: "dd", "dt"**

Run these steps:

1. Set the frameset-ok flag[p1387] to "not ok".
2. Initialize *node* to be the current node[p1384] (the bottommost node of the stack).
3. *Loop*: If *node* is a dd[p258] element, then run these substeps:
    1. Generate implied end tags[p1423], except for dd[p258] elements.
    2. If the current node[p1384] is not a dd[p258] element, then this is a parse error[p1370].
    3. Pop elements from the stack of open elements[p1383] until a dd[p258] element has been popped from the stack.
    4. Jump to the step labeled *done* below.
4. If *node* is a dt[p257] element, then run these substeps:
    1. Generate implied end tags[p1423], except for dt[p257] elements.
    2. If the current node[p1384] is not a dt[p257] element, then this is a parse error[p1370].
    3. Pop elements from the stack of open elements[p1383] until a dt[p257] element has been popped from the stack.
    4. Jump to the step labeled *done* below.
5. If *node* is in the special[p1384] category, but is not an address[p230], div[p266], or p[p238] element, then jump to the step labeled *done* below.
6. Otherwise, set *node* to the previous entry in the stack of open elements[p1383] and return to the step labeled *loop*.
7. *Done*: If the stack of open elements[p1383] has a p element in button scope[p1385], then close a p element[p1442].
8. Finally, insert an HTML element[p1420] for the token.

↪ **A start tag whose tag name is "plaintext"**

If the stack of open elements[p1383] has a p element in button scope[p1385], then close a p element[p1442].

Insert an HTML element[p1420] for the token.

Switch the tokenizer to the PLAINTEXT state[p1389].

Note

Once a start tag with the tag name "plaintext" has been seen, all remaining tokens will be character tokens (and a final end-of-file token) because there is no way to switch the tokenizer out of the PLAINTEXT state[p1389]. However, as the tree builder remains in its existing insertion mode, it might reconstruct the active formatting elements[p1386] while processing those character tokens. This means that the parser can insert other elements into the `plaintext`[p1530] element.

↪ **A start tag whose tag name is "button"**

1. If the stack of open elements[p1383] has a `button` element in scope[p1384], then run these substeps:
    1. Parse error[p1370].
    2. Generate implied end tags[p1423].
    3. Pop elements from the stack of open elements[p1383] until a `button`[p585] element has been popped from the stack.
2. Reconstruct the active formatting elements[p1386], if any.
3. Insert an HTML element[p1420] for the token.
4. Set the frameset-ok flag[p1387] 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", "search", "section", "select", "summary", "ul"**

If the stack of open elements[p1383] does not have an element in scope[p1384] that is an HTML element[p46] with the same tag name as that of the token, then this is a parse error[p1370]; ignore the token.

Otherwise, run these steps:

1. Generate implied end tags[p1423].
2. If the current node[p1384] is not an HTML element[p46] with the same tag name as that of the token, then this is a parse error[p1370].
3. Pop elements from the stack of open elements[p1383] until an HTML element[p46] with the same tag name as the token has been popped from the stack.

↪ **An end tag whose tag name is "form"**

If the parser is not parsing template contents[p1386]:

1. Let *node* be the element that the `form` element pointer[p1386] is set to, or null if it is not set to an element.
2. Set the `form` element pointer[p1386] to null.
3. If *node* is null or if the stack of open elements[p1383] does not have *node* in scope[p1384], then this is a parse error[p1370]; return and ignore the token.
4. Generate implied end tags[p1423].
5. If the current node[p1384] is not *node*, then this is a parse error[p1370].
6. Remove *node* from the stack of open elements[p1383].

Otherwise:

1. If the stack of open elements[p1383] does not have a `form` element in scope[p1384], then this is a parse error[p1370]; return and ignore the token.
2. Generate implied end tags[p1423].
3. If the current node[p1384] is not a `form`[p532] element, then this is a parse error[p1370].

4. Pop elements from the stack of open elements[p1383] until a form[p532] element has been popped from the stack.

↪ **An end tag whose tag name is "p"**

If the stack of open elements[p1383] does not have a p element in button scope[p1385], then this is a parse error[p1370]; insert an HTML element[p1420] for a "p" start tag token with no attributes.

Close a p element[p1442].

↪ **An end tag whose tag name is "li"**

If the stack of open elements[p1383] does not have an li element in list item scope[p1385], then this is a parse error[p1370]; ignore the token.

Otherwise, run these steps:

1. Generate implied end tags[p1423], except for li[p251] elements.
2. If the current node[p1384] is not an li[p251] element, then this is a parse error[p1370].
3. Pop elements from the stack of open elements[p1383] until an li[p251] element has been popped from the stack.

↪ **An end tag whose tag name is one of: "dd", "dt"**

If the stack of open elements[p1383] does not have an element in scope[p1384] that is an HTML element[p46] with the same tag name as that of the token, then this is a parse error[p1370]; ignore the token.

Otherwise, run these steps:

1. Generate implied end tags[p1423], except for HTML elements[p46] with the same tag name as the token.
2. If the current node[p1384] is not an HTML element[p46] with the same tag name as that of the token, then this is a parse error[p1370].
3. Pop elements from the stack of open elements[p1383] until an HTML element[p46] with the same tag name as the token has been popped from the stack.

↪ **An end tag whose tag name is one of: "h1", "h2", "h3", "h4", "h5", "h6"**

If the stack of open elements[p1383] does not have an element in scope[p1384] that is an HTML element[p46] and whose tag name is one of "h1", "h2", "h3", "h4", "h5", or "h6", then this is a parse error[p1370]; ignore the token.

Otherwise, run these steps:

1. Generate implied end tags[p1423].
2. If the current node[p1384] is not an HTML element[p46] with the same tag name as that of the token, then this is a parse error[p1370].
3. Pop elements from the stack of open elements[p1383] until an HTML element[p46] 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[p1385] contains an a[p267] element between the end of the list and the last marker[p1385] on the list (or the start of the list if there is no marker[p1385] on the list), then this is a parse error[p1370]; run the adoption agency algorithm[p1442] for the token, then remove that element from the list of active formatting elements[p1385] and the stack of open elements[p1383] if the adoption agency algorithm[p1442] didn't already remove it (it might not have if the element is not in table scope[p1385]).

Example

In the non-conforming stream `<a href="a">a<table><a href="b">b</table>x`, the first a[p267] 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[p267] 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[p267] element). The result is that the two a[p267] elements are indirectly nested inside each other — non-conforming markup will often result in non-conforming DOMs when parsed.

Reconstruct the active formatting elements[p1386], if any.

Insert an HTML element[p1420] for the token. Push onto the list of active formatting elements[p1385] 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[p1386], if any.

Insert an HTML element[p1420] for the token. Push onto the list of active formatting elements[p1385] that element.

↪ **A start tag whose tag name is "nobr"**

Reconstruct the active formatting elements[p1386], if any.

If the stack of open elements[p1383] has a nobr element in scope[p1384], then this is a parse error[p1370]; run the adoption agency algorithm[p1442] for the token, then once again reconstruct the active formatting elements[p1386], if any.

Insert an HTML element[p1420] for the token. Push onto the list of active formatting elements[p1385] that element.

↪ **An end tag whose tag name is one of: "a", "b", "big", "code", "em", "font", "i", "nobr", "s", "small", "strike", "strong", "tt", "u"**

Run the adoption agency algorithm[p1442] for the token.

↪ **A start tag whose tag name is one of: "applet", "marquee", "object"**

Reconstruct the active formatting elements[p1386], if any.

Insert an HTML element[p1420] for the token.

Insert a marker[p1385] at the end of the list of active formatting elements[p1385].

Set the frameset-ok flag[p1387] to "not ok".

↪ **An end tag token whose tag name is one of: "applet", "marquee", "object"**

If the stack of open elements[p1383] does not have an element in scope[p1384] that is an HTML element[p46] with the same tag name as that of the token, then this is a parse error[p1370]; ignore the token.

Otherwise, run these steps:

1. Generate implied end tags[p1423].
2. If the current node[p1384] is not an HTML element[p46] with the same tag name as that of the token, then this is a parse error[p1370].
3. Pop elements from the stack of open elements[p1383] until an HTML element[p46] with the same tag name as the token has been popped from the stack.
4. Clear the list of active formatting elements up to the last marker[p1386].

↪ **A start tag whose tag name is "table"**

If the Document[p135] is *not* set to quirks mode, and the stack of open elements[p1383] has a p element in button scope[p1385], then close a p element[p1442].

Insert an HTML element[p1420] for the token.

Set the frameset-ok flag[p1387] to "not ok".

Switch the insertion mode[p1382] to "in table[p1445]".

↪ **An end tag whose tag name is "br"**

Parse error[p1370]. Drop the attributes from the token, and act as described in the next entry; i.e. act as if this was a "br" start tag token with no attributes, rather than the end tag token that it actually is.

↪ **A start tag whose tag name is one of: "area", "br", "embed", "img", "keygen", "wbr"**

Reconstruct the active formatting elements[p1386], if any.

Insert an HTML element[p1420] for the token. Immediately pop the current node[p1384] off the stack of open elements[p1383].

Acknowledge the token's *self-closing flag*[p1387], if it is set.

Set the frameset-ok flag[p1387] to "not ok".

↪ **A start tag whose tag name is "input"**

If the parser's fragment context element[p1386] is a `select`[p590] element (fragment case[p1473]):

1. Parse error[p1370].
2. Ignore the token.
3. Return.

If the stack of open elements[p1383] has a `select` element in scope[p1384]:

1. Parse error[p1370].
2. Pop elements from the stack of open elements[p1383] until a `select`[p590] element has been popped from the stack.

Reconstruct the active formatting elements[p1386], if any.

Insert an HTML element[p1420] for the token. Immediately pop the current node[p1384] off the stack of open elements[p1383].

Acknowledge the token's *self-closing flag*[p1387], 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 "`hidden`", then set the frameset-ok flag[p1387] to "not ok".

↪ **A start tag whose tag name is one of: "param", "source", "track"**

Insert an HTML element[p1420] for the token. Immediately pop the current node[p1384] off the stack of open elements[p1383].

Acknowledge the token's *self-closing flag*[p1387], if it is set.

↪ **A start tag whose tag name is "hr"**

If the stack of open elements[p1383] has a `p` element in button scope[p1385], then close a `p` element[p1442].

If the stack of open elements[p1383] has a `select` element in scope[p1384]:

1. Generate implied end tags[p1423].
2. If the stack of open elements[p1383] has an `option` element in scope[p1384] or has an `optgroup` element in scope[p1384], then this is a parse error[p1370].

Insert an HTML element[p1420] for the token. Immediately pop the current node[p1384] off the stack of open elements[p1383].

Acknowledge the token's *self-closing flag*[p1387], if it is set.

Set the frameset-ok flag[p1387] to "not ok".

↪ **A start tag whose tag name is "image"**

Parse error[p1370]. 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[p1420] for the token.
2. If the next token[p1417] 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`[p606] elements are ignored as an authoring convenience.)
3. Switch the tokenizer to the RCDATA state[p1388].
4. Set the original insertion mode[p1382] to the current insertion mode[p1382].
5. Set the frameset-ok flag[p1387] to "not ok".
6. Switch the insertion mode[p1382] to "text[p1443]".

↪ **A start tag whose tag name is "xmp"**

If the stack of open elements[p1383] has a p element in button scope[p1385], then close a p element[p1442].

Reconstruct the active formatting elements[p1386], if any.

Set the frameset-ok flag[p1387] to "not ok".

Follow the generic raw text element parsing algorithm[p1423].

↪ **A start tag whose tag name is "iframe"**

Set the frameset-ok flag[p1387] to "not ok".

Follow the generic raw text element parsing algorithm[p1423].

↪ **A start tag whose tag name is "noembed"**

↪ **A start tag whose tag name is "noscript", if scripting mode[p1387] is not Disabled[p1387]**

Follow the generic raw text element parsing algorithm[p1423].

↪ **A start tag whose tag name is "select"**

If the parser's fragment context element[p1386] is a `select`[p590] element (fragment case[p1473]):

1. Parse error[p1370].
2. Ignore the token.

Otherwise, if the stack of open elements[p1383] has a `select` element in scope[p1384]:

1. Parse error[p1370].
2. Ignore the token.
3. Pop elements from the stack of open elements[p1383] until a `select`[p590] element has been popped from the stack.

Otherwise:

1. Reconstruct the active formatting elements[p1386], if any.
2. Insert an HTML element[p1420] for the token.
3. Set the frameset-ok flag[p1387] to "not ok".

↪ **A start tag whose tag name is "option"**

If the stack of open elements[p1383] has a `select` element in scope[p1384]:

1. Generate implied end tags[p1423] except for `optgroup`[p600] elements.
2. If the stack of open elements[p1383] has an `option` element in scope[p1384], then this is a parse error[p1370].

Otherwise:

1. If the current node[p1384] is an `option`[p601] element, then pop the current node[p1384] off the stack of open elements[p1383].

Reconstruct the active formatting elements[p1386], if any.

Insert an HTML element[p1420] for the token.

↪ **A start tag whose tag name is "optgroup"**

If the stack of open elements[p1383] has a `select` element in scope[p1384]:

1. Generate implied end tags[p1423].
2. If the stack of open elements[p1383] has an `option` element in scope[p1384] or has an `optgroup` element in scope[p1384], then this is a parse error[p1370].

Otherwise, if the current node[p1384] is an `option`[p601] element, then pop the current node[p1384] from the stack of open elements[p1383].

Reconstruct the active formatting elements[p1386], if any.

Insert an HTML element[p1420] for the token.

↪ **A start tag whose tag name is one of: "rb", "rtc"**

If the stack of open elements[p1383] has a ruby element in scope[p1384], then generate implied end tags[p1423]. If the current node[p1384] is not now a ruby[p281] element, this is a parse error[p1370].

Insert an HTML element[p1420] for the token.

↪ **A start tag whose tag name is one of: "rp", "rt"**

If the stack of open elements[p1383] has a ruby element in scope[p1384], then generate implied end tags[p1423], except for rtc[p1531] elements. If the current node[p1384] is not now a rtc[p1531] element or a ruby[p281] element, this is a parse error[p1370].

Insert an HTML element[p1420] for the token.

↪ **A start tag whose tag name is "math"**

Reconstruct the active formatting elements[p1386], if any.

Adjust MathML attributes[p1420] for the token. (This fixes the case of MathML attributes that are not all lowercase.)

Adjust foreign attributes[p1421] for the token. (This fixes the use of namespaced attributes, in particular XLink.)

Insert a foreign element[p1420] for the token, with MathML namespace and false.

If the token has its *self-closing flag*[p1387] set, pop the current node[p1384] off the stack of open elements[p1383] and acknowledge the token's *self-closing flag*[p1387].

↪ **A start tag whose tag name is "svg"**

Reconstruct the active formatting elements[p1386], if any.

Adjust SVG attributes[p1420] for the token. (This fixes the case of SVG attributes that are not all lowercase.)

Adjust foreign attributes[p1421] for the token. (This fixes the use of namespaced attributes, in particular XLink in SVG.)

Insert a foreign element[p1420] for the token, with SVG namespace and false.

If the token has its *self-closing flag*[p1387] set, pop the current node[p1384] off the stack of open elements[p1383] and acknowledge the token's *self-closing flag*[p1387].

↪ **A start tag whose tag name is one of: "caption", "col", "colgroup", "frame", "head", "tbody", "td", "tfoot", "th", "thead", "tr"**

Parse error[p1370]. Ignore the token.

↪ **Any other start tag**

Reconstruct the active formatting elements[p1386], if any.

Insert an HTML element[p1420] for the token.

Note

This element will be an ordinary[p1384] element. With one exception: if scripting mode[p1387] is Disabled[p1387], it can also be a noscript[p703] *element.*

↪ **Any other end tag**

Run these steps:

1. Initialize *node* to be the current node[p1384] (the bottommost node of the stack).
2. *Loop*: If *node* is an HTML element[p46] with the same tag name as the token:
    1. Generate implied end tags[p1423], except for HTML elements[p46] with the same tag name as the token.
    2. If *node* is not the current node[p1384], then this is a parse error[p1370].
    3. Pop all the nodes from the current node[p1384] up to *node*, including *node*, then stop these steps.

3. Otherwise, if *node* is in the special[p1384] category, then this is a parse error[p1370]; ignore the token, and return.
4. Set *node* to the previous entry in the stack of open elements[p1383].
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[p1423], except for p[p238] elements.
2. If the current node[p1384] is not a p[p238] element, then this is a parse error[p1370].
3. Pop elements from the stack of open elements[p1383] until a p[p238] 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[p1384] is an HTML element[p46] whose tag name is *subject*, and the current node[p1384] is not in the list of active formatting elements[p1385], then pop the current node[p1384] off the stack of open elements[p1383] and return.
3. Let *outerLoopCounter* be 0.
4. While true:
    1. If *outerLoopCounter* is greater than or equal to 8, then return.
    2. Increment *outerLoopCounter* by 1.
    3. Let *formattingElement* be the last element in the list of active formatting elements[p1385] that:
        - is between the end of the list and the last marker[p1385] 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 act as described in the "any other end tag" entry above and return.
    4. If *formattingElement* is not in the stack of open elements[p1383], then this is a parse error[p1370]; remove the element from the list, and return.
    5. If *formattingElement* is in the stack of open elements[p1383], but the element is not in scope[p1384], then this is a parse error[p1370]; return.
    6. If *formattingElement* is not the current node[p1384], this is a parse error[p1370]. (But do not return.)
    7. Let *furthestBlock* be the topmost node in the stack of open elements[p1383] that is lower in the stack than *formattingElement*, and is an element in the special[p1384] category. There might not be one.
    8. If there is no *furthestBlock*, then the UA must first pop all the nodes from the bottom of the stack of open elements[p1383], from the current node[p1384] up to and including *formattingElement*, then remove *formattingElement* from the list of active formatting elements[p1385], and finally return.
    9. Let *commonAncestor* be the element immediately above *formattingElement* in the stack of open elements[p1383].
    10. Let a bookmark note the position of *formattingElement* in the list of active formatting elements[p1385] relative to the elements on either side of it in the list.
    11. Let *node* and *lastNode* be *furthestBlock*.
    12. Let *innerLoopCounter* be 0.
    13. While true:
        1. Increment *innerLoopCounter* by 1.
        2. Let *node* be the element immediately above *node* in the stack of open elements[p1383], or if *node* is no longer in the stack of open elements[p1383] (e.g. because it got removed by this algorithm), the element that was immediately above *node* in the stack of open elements[p1383] before *node* was removed.

3. If *node* is *formattingElement*, then break.

4. If *innerLoopCounter* is greater than 3 and *node* is in the list of active formatting elements[p1385], then remove *node* from the list of active formatting elements[p1385].

5. If *node* is not in the list of active formatting elements[p1385], then remove *node* from the stack of open elements[p1383] and continue.

6. Create an element for the token[p1418] for which the element *node* was created, in the HTML namespace, with *commonAncestor* as the intended parent; replace the entry for *node* in the list of active formatting elements[p1385] with an entry for the new element, replace the entry for *node* in the stack of open elements[p1383] with an entry for the new element, and let *node* be the new element.

7. If *lastNode* is *furthestBlock*, then move the aforementioned bookmark to be immediately after the new *node* in the list of active formatting elements[p1385].

8. Append *lastNode* to *node*.

9. Set *lastNode* to *node*.

14. Let (*target*, *refNode*) be the adjusted insertion location[p1419] given (*commonAncestor*, null).

15. If *lastNode*'s parent is non-null, then remove *lastNode*.

16. If all of the following are true:

    - *lastNode*'s parent is null;
    - *lastNode* is not a host-including inclusive ancestor of *target*;
    - *target* is not a `Document`[p135] node, or it does not have an element child; and
    - *refNode* is null or its parent is *target*,

    then:

    1. Assert: ensure pre-insert validity given *lastNode*, *target*, *refNode*, and « » does not throw.
    2. Insert *lastNode* into *target* before *refNode*.

17. Create an element for the token[p1418] for which *formattingElement* was created, in the HTML namespace, with *furthestBlock* as the intended parent.

18. Take all of the child nodes of *furthestBlock* and append them to the element created in the last step.

19. Append that new element to *furthestBlock*.

20. Remove *formattingElement* from the list of active formatting elements[p1385], and insert the new element into the list of active formatting elements[p1385] at the position of the aforementioned bookmark.

21. Remove *formattingElement* from the stack of open elements[p1383], and insert the new element into the stack of open elements[p1383] immediately below the position of *furthestBlock* 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 §[p1443]

When the user agent is to apply the rules for the "text[p1443]" insertion mode[p1382], the user agent must handle the token as follows:

↪ A character token

Insert the token's character[p1422].

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[p1370].

If the current node[p1384] is a `script`[p685] element, then set its already started[p692] to true.

Pop the current node[p1384] off the stack of open elements[p1383].

Switch the insertion mode[p1382] to the original insertion mode[p1382] and reprocess the token.

↪ **An end tag whose tag name is "script"**

If the active speculative HTML parser[p1459] is null and the JavaScript execution context stack is empty, then perform a microtask checkpoint[p1200].

Let *script* be the current node[p1384] (which will be a `script`[p685] element).

Pop the current node[p1384] off the stack of open elements[p1383].

Switch the insertion mode[p1382] to the original insertion mode[p1382].

Let the *old insertion point* have the same value as the current insertion point[p1382]. Let the insertion point[p1382] be just before the next input character[p1382].

Increment the parser's script nesting level[p1370] by one.

If the active speculative HTML parser[p1459] is null, then prepare the script element[p694] *script*. This might cause some script to execute, which might cause new characters to be inserted into the tokenizer[p1223], and might cause the tokenizer to output more tokens, resulting in a reentrant invocation of the parser[p1369].

Decrement the parser's script nesting level[p1370] by one. If the parser's script nesting level[p1370] is zero, then set the parser pause flag[p1370] to false.

Let the insertion point[p1382] have the value of the *old insertion point*. (In other words, restore the insertion point[p1382] to its previous value. This value might be the "undefined" value.)

At this stage, if the pending parsing-blocking script[p698] is not null:

↪ **If the script nesting level[p1370] is not zero:**

Set the parser pause flag[p1370] 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[p1369], say from a call to `document.write()`[p1223].

↪ **Otherwise:**

While the pending parsing-blocking script[p698] is not null:

1. Let *the script* be the pending parsing-blocking script[p698].
2. Set the pending parsing-blocking script[p698] to null.
3. Start the speculative HTML parser[p1460] for this instance of the HTML parser.
4. Block the tokenizer[p1387] for this instance of the HTML parser[p1368], such that the event loop[p1193] will not run tasks[p1194] that invoke the tokenizer[p1387].
5. If the parser's `Document`[p135] has a style sheet that is blocking scripts[p212] or *the script*'s ready to be parser-executed[p692] is false: spin the event loop[p1201] until the parser's `Document`[p135] has no style sheet that is blocking scripts[p212] and *the script*'s ready to be parser-executed[p692] becomes true.
6. If this parser has been aborted[p1459] in the meantime, return.

Note

This could happen if, e.g., while the spin the event loop[p1201] algorithm is running, the `Document`[p135] *gets*

destroyed[p1116], *or the* `document.open()`[p1221] *method gets invoked on the* `Document`[p135].

7. Stop the speculative HTML parser[p1460] for this instance of the HTML parser.
8. Unblock the tokenizer[p1387] for this instance of the HTML parser[p1368], such that tasks[p1194] that invoke the tokenizer[p1387] can again be run.
9. Let the insertion point[p1382] be just before the next input character[p1382].
10. Increment the parser's script nesting level[p1370] by one (it should be zero before this step, so this sets it to one).
11. Execute the script element[p698] *the script*.
12. Decrement the parser's script nesting level[p1370] by one. If the parser's script nesting level[p1370] is zero (which it always should be at this point), then set the parser pause flag[p1370] to false.
13. Let the insertion point[p1382] be undefined again.

↪ **Any other end tag**

Pop the current node[p1384] off the stack of open elements[p1383].

Switch the insertion mode[p1382] to the original insertion mode[p1382].

13.2.6.4.9 The "in table" insertion mode §[p1445]

When the user agent is to apply the rules for the "in table[p1445]" insertion mode[p1382], the user agent must handle the token as follows:

↪ **A character token, if the current node[p1384] is `table`[p496], `tbody`[p507], `template`[p705], `tfoot`[p509], `thead`[p508], or `tr`[p510] element**

Let the ***pending table character tokens*** be an empty list of tokens.

Set the original insertion mode[p1382] to the current insertion mode[p1382].

Switch the insertion mode[p1382] to "in table text[p1447]" and reprocess the token.

↪ **A comment token**

Insert a comment[p1423].

↪ **A processing instruction token**

Insert a processing instruction[p1423].

↪ **A DOCTYPE token**

Parse error[p1370]. Ignore the token.

↪ **A start tag whose tag name is "caption"**

Clear the stack back to a table context[p1447]. (See below.)

Insert a marker[p1385] at the end of the list of active formatting elements[p1385].

Insert an HTML element[p1420] for the token, then switch the insertion mode[p1382] to "in caption[p1447]".

↪ **A start tag whose tag name is "colgroup"**

Clear the stack back to a table context[p1447]. (See below.)

Insert an HTML element[p1420] for the token, then switch the insertion mode[p1382] to "in column group[p1448]".

↪ **A start tag whose tag name is "col"**

Clear the stack back to a table context[p1447]. (See below.)

Insert an HTML element[p1420] for a "colgroup" start tag token with no attributes, then switch the insertion mode[p1382] to "in column group[p1448]".

Reprocess the current token.

↪ **A start tag whose tag name is one of: "tbody", "tfoot", "thead"**

Clear the stack back to a table context[p1447]. (See below.)

Insert an HTML element[p1420] for the token, then switch the insertion mode[p1382] to "in table body[p1449]".

↪ **A start tag whose tag name is one of: "td", "th", "tr"**

Clear the stack back to a table context[p1447]. (See below.)

Insert an HTML element[p1420] for a "tbody" start tag token with no attributes, then switch the insertion mode[p1382] to "in table body[p1449]".

Reprocess the current token.

↪ **A start tag whose tag name is "table"**

Parse error[p1370].

If the stack of open elements[p1383] does not have a `table` element in table scope[p1385], ignore the token.

Otherwise:

1. Pop elements from this stack until a `table`[p496] element has been popped from the stack.
2. Reset the insertion mode appropriately[p1383].
3. Reprocess the token.

↪ **An end tag whose tag name is "table"**

If the stack of open elements[p1383] does not have a `table` element in table scope[p1385], this is a parse error[p1370]; ignore the token.

Otherwise:

1. Pop elements from this stack until a `table`[p496] element has been popped from the stack.
2. Reset the insertion mode appropriately[p1383].

↪ **An end tag whose tag name is one of: "body", "caption", "col", "colgroup", "html", "tbody", "td", "tfoot", "th", "thead", "tr"**

Parse error[p1370]. 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[p1382] the "in head[p1427]" insertion mode[p1382].

↪ **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 "`hidden`", then act as described in the "anything else" entry below.

Otherwise:

1. Parse error[p1370].
2. Insert an HTML element[p1420] for the token.
3. Pop that `input`[p539] element off the stack of open elements[p1383].
4. Acknowledge the token's *self-closing flag*[p1387], if it is set.

↪ **A start tag whose tag name is "form"**

Parse error[p1370].

If the `form` element pointer[p1386] is not null, and the parser is not parsing template contents[p1386], then ignore the token.

Otherwise:

1. Insert an HTML element[p1420] for the token, and, if the parser is not parsing template contents[p1386], set the `form` element pointer[p1386] to point to the element created.

2. 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:

1. Generate implied end tags.
2. Now, if the current node is not a caption element, then this is a parse error.
3. Pop elements from this stack until a caption element has been popped from the stack.
4. Clear the list of active formatting elements up to the last marker.
5. Switch the insertion mode to "in 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[p1383] does not have a `caption` element in table scope[p1385], this is a parse error[p1370]; ignore the token. (fragment case[p1473])

Otherwise:

1. Generate implied end tags[p1423].
2. Now, if the current node[p1384] is not a `caption`[p504] element, then this is a parse error[p1370].
3. Pop elements from this stack until a `caption`[p504] element has been popped from the stack.
4. Clear the list of active formatting elements up to the last marker[p1386].
5. Switch the insertion mode[p1382] to "in table[p1445]".
6. Reprocess the token.

↪ **An end tag whose tag name is one of: "body", "col", "colgroup", "html", "tbody", "td", "tfoot", "th", "thead", "tr"**

Parse error[p1370]. Ignore the token.

↪ **Anything else**

Process the token using the rules for[p1382] the "in body[p1432]" insertion mode[p1382].

13.2.6.4.12 The "in column group" insertion mode §[p1448]

When the user agent is to apply the rules for the "in column group[p1448]" insertion mode[p1382], 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[p1422].

↪ **A comment token**

Insert a comment[p1423].

↪ **A processing instruction token**

Insert a processing instruction[p1423].

↪ **A DOCTYPE token**

Parse error[p1370]. Ignore the token.

↪ **A start tag whose tag name is "html"**

Process the token using the rules for[p1382] the "in body[p1432]" insertion mode[p1382].

↪ **A start tag whose tag name is "col"**

Insert an HTML element[p1420] for the token. Immediately pop the current node[p1384] off the stack of open elements[p1383].

Acknowledge the token's *self-closing flag*[p1387], if it is set.

↪ **An end tag whose tag name is "colgroup"**

If the current node[p1384] is not a `colgroup`[p505] element, then this is a parse error[p1370]; ignore the token.

Otherwise, pop the current node[p1384] from the stack of open elements[p1383]. Switch the insertion mode[p1382] to "in table[p1445]".

↪ **An end tag whose tag name is "col"**

Parse error[p1370]. Ignore the token.

↪ **A start tag whose tag name is "template"**

↪ **An end tag whose tag name is "template"**

Process the token using the rules for[p1382] the "in head[p1427]" insertion mode[p1382].

↪ **An end-of-file token**

Process the token using the rules for[p1382] the "in body[p1432]" insertion mode[p1382].

↪ **Anything else**

If the current node[p1384] is not a `colgroup`[p505] element, then this is a parse error[p1370]; ignore the token.

Otherwise, pop the current node[p1384] from the stack of open elements[p1383].

Switch the insertion mode[p1382] to "in table[p1445]".

Reprocess the token.

13.2.6.4.13 The "in table body" insertion mode §[p1449]

When the user agent is to apply the rules for the "in table body[p1449]" insertion mode[p1382], 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[p1449]. (See below.)

Insert an HTML element[p1420] for the token, then switch the insertion mode[p1382] to "in row[p1450]".

↪ **A start tag whose tag name is one of: "th", "td"**

Parse error[p1370].

Clear the stack back to a table body context[p1449]. (See below.)

Insert an HTML element[p1420] for a "tr" start tag token with no attributes, then switch the insertion mode[p1382] to "in row[p1450]".

Reprocess the current token.

↪ **An end tag whose tag name is one of: "tbody", "tfoot", "thead"**

If the stack of open elements[p1383] does not have an element in table scope[p1385] that is an HTML element[p46] with the same tag name as the token, this is a parse error[p1370]; ignore the token.

Otherwise:

1. Clear the stack back to a table body context[p1449]. (See below.)
2. Pop the current node[p1384] from the stack of open elements[p1383]. Switch the insertion mode[p1382] to "in table[p1445]".

↪ **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[p1383] does not have a `tbody`, `thead`, or `tfoot` element in table scope[p1385], this is a parse error[p1370]; ignore the token.

Otherwise:

1. Clear the stack back to a table body context[p1449]. (See below.)
2. Pop the current node[p1384] from the stack of open elements[p1383]. Switch the insertion mode[p1382] to "in table[p1445]".
3. Reprocess the token.

↪ **An end tag whose tag name is one of: "body", "caption", "col", "colgroup", "html", "td", "th", "tr"**

Parse error[p1370]. Ignore the token.

↪ **Anything else**

Process the token using the rules for[p1382] the "in table[p1445]" insertion mode[p1382].

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[p1384] is not a `tbody`[p507], `tfoot`[p509], `thead`[p508], `template`[p705], or `html`[p179] element, pop elements from the stack of open elements[p1383].

Note

The current node[p1384] being an html[p179] element after this process is a fragment case[p1473].

13.2.6.4.14 The "in row" insertion mode §[p1450]

When the user agent is to apply the rules for the "in row[p1450]" insertion mode[p1382], 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[p1450]. (See below.)

Insert an HTML element[p1420] for the token, then switch the insertion mode[p1382] to "in cell[p1451]".

Insert a marker[p1385] at the end of the list of active formatting elements[p1385].

↪ **An end tag whose tag name is "tr"**

If the stack of open elements[p1383] does not have a tr element in table scope[p1385], this is a parse error[p1370]; ignore the token.

Otherwise:

1. Clear the stack back to a table row context[p1450]. (See below.)
2. Pop the current node[p1384] (which will be a tr[p510] element) from the stack of open elements[p1383]. Switch the insertion mode[p1382] to "in table body[p1449]".

↪ **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[p1383] does not have a tr element in table scope[p1385], this is a parse error[p1370]; ignore the token.

Otherwise:

1. Clear the stack back to a table row context[p1450]. (See below.)
2. Pop the current node[p1384] (which will be a tr[p510] element) from the stack of open elements[p1383]. Switch the insertion mode[p1382] to "in table body[p1449]".
3. Reprocess the token.

↪ **An end tag whose tag name is one of: "tbody", "tfoot", "thead"**

If the stack of open elements[p1383] does not have an element in table scope[p1385] that is an HTML element[p46] with the same tag name as the token, this is a parse error[p1370]; ignore the token.

If the stack of open elements[p1383] does not have a tr element in table scope[p1385], ignore the token.

Otherwise:

1. Clear the stack back to a table row context[p1450]. (See below.)
2. Pop the current node[p1384] (which will be a tr[p510] element) from the stack of open elements[p1383]. Switch the insertion mode[p1382] to "in table body[p1449]".
3. Reprocess the token.

↪ **An end tag whose tag name is one of: "body", "caption", "col", "colgroup", "html", "td", "th"**

Parse error[p1370]. Ignore the token.

↪ **Anything else**

Process the token using the rules for[p1382] the "in table[p1445]" insertion mode[p1382].

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[p1384] is not a tr[p510], template[p705], or html[p179] element, pop elements from the stack of open elements[p1383].

Note

The current node[p1384] being an html[p179] element after this process is a fragment case[p1473].

13.2.6.4.15 The "in cell" insertion mode §[p1451]

When the user agent is to apply the rules for the "in cell[p1451]" insertion mode[p1382], 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[p1383] does not have an element in table scope[p1385] that is an HTML element[p46] with the same tag name as that of the token, then this is a parse error[p1370]; ignore the token.

Otherwise:

1. Generate implied end tags[p1423].
2. Now, if the current node[p1384] is not an HTML element[p46] with the same tag name as the token, then this is a parse error[p1370].
3. Pop elements from the stack of open elements[p1383] until an HTML element[p46] with the same tag name as the token has been popped from the stack.
4. Clear the list of active formatting elements up to the last marker[p1386].
5. Switch the insertion mode[p1382] to "in row[p1450]".

↪ **A start tag whose tag name is one of: "caption", "col", "colgroup", "tbody", "td", "tfoot", "th", "thead", "tr"**

Assert: the stack of open elements[p1383] has a td or th element in table scope[p1385].

Close the cell[p1451] (see below) and reprocess the token.

↪ **An end tag whose tag name is one of: "body", "caption", "col", "colgroup", "html"**

Parse error[p1370]. Ignore the token.

↪ **An end tag whose tag name is one of: "table", "tbody", "tfoot", "thead", "tr"**

If the stack of open elements[p1383] does not have an element in table scope[p1385] that is an HTML element[p46] with the same tag name as that of the token, then this is a parse error[p1370]; ignore the token.

Otherwise, close the cell[p1451] (see below) and reprocess the token.

↪ **Anything else**

Process the token using the rules for[p1382] the "in body[p1432]" insertion mode[p1382].

Where the steps above say to **close the cell**, they mean to run the following algorithm:

1. Generate implied end tags[p1423].
2. If the current node[p1384] is not now a td[p512] element or a th[p513] element, then this is a parse error[p1370].
3. Pop elements from the stack of open elements[p1383] until a td[p512] element or a th[p513] element has been popped from the stack.
4. Clear the list of active formatting elements up to the last marker[p1386].
5. Switch the insertion mode[p1382] to "in row[p1450]".

Note

The stack of open elements[p1383] cannot have both a td[p512] and a th[p513] element in table scope[p1385] at the same time, nor can it have neither when the close the cell[p1451] algorithm is invoked.

13.2.6.4.16 The "in template" insertion mode §[p1452]

When the user agent is to apply the rules for the "in template[p1452]" insertion mode[p1382], the user agent must handle the token as follows:

↪ **A character token**

↪ **A comment token**

↪ **A processing instruction token**

↪ **A DOCTYPE token**

Process the token using the rules for[p1382] the "in body[p1432]" insertion mode[p1382].

↪ **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[p1382] the "in head[p1427]" insertion mode[p1382].

↪ **A start tag whose tag name is one of: "caption", "colgroup", "tbody", "tfoot", "thead"**

Pop the current template insertion mode[p1383] off the stack of template insertion modes[p1383].

Push "in table[p1445]" onto the stack of template insertion modes[p1383] so that it is the new current template insertion mode[p1383].

Switch the insertion mode[p1382] to "in table[p1445]", and reprocess the token.

↪ **A start tag whose tag name is "col"**

Pop the current template insertion mode[p1383] off the stack of template insertion modes[p1383].

Push "in column group[p1448]" onto the stack of template insertion modes[p1383] so that it is the new current template insertion mode[p1383].

Switch the insertion mode[p1382] to "in column group[p1448]", and reprocess the token.

↪ **A start tag whose tag name is "tr"**

Pop the current template insertion mode[p1383] off the stack of template insertion modes[p1383].

Push "in table body[p1449]" onto the stack of template insertion modes[p1383] so that it is the new current template insertion mode[p1383].

Switch the insertion mode[p1382] to "in table body[p1449]", and reprocess the token.

↪ **A start tag whose tag name is one of: "td", "th"**

Pop the current template insertion mode[p1383] off the stack of template insertion modes[p1383].

Push "in row[p1450]" onto the stack of template insertion modes[p1383] so that it is the new current template insertion mode[p1383].

Switch the insertion mode[p1382] to "in row[p1450]", and reprocess the token.

↪ **Any other start tag**

Pop the current template insertion mode[p1383] off the stack of template insertion modes[p1383].

Push "in body[p1432]" onto the stack of template insertion modes[p1383] so that it is the new current template insertion mode[p1383].

Switch the insertion mode[p1382] to "in body[p1432]", and reprocess the token.

↪ **Any other end tag**

Parse error[p1370]. Ignore the token.

↪ **An end-of-file token**

If there is no `template`[p705] element on the stack of open elements[p1383], then stop parsing[p1458]. (fragment case[p1473])

Otherwise, this is a parse error[p1370].

Pop elements from the stack of open elements[p1383] until a `template`[p705] element has been popped from the stack.

Clear the list of active formatting elements up to the last marker[p1386].

Pop the current template insertion mode[p1383] off the stack of template insertion modes[p1383].

Reset the insertion mode appropriately[p1383].

Reprocess the token.

13.2.6.4.17 The "after body" insertion mode §[p14 53]

When the user agent is to apply the rules for the "after body[p1453]" insertion mode[p1382], the user agent must handle the token as follows:

↪ **A character token that is one of U+0009 CHARACTER TABULATION, U+000A LINE FEED (LF), U+000C FORM FEED (FF), U+000D CARRIAGE RETURN (CR), or U+0020 SPACE**

Process the token using the rules for[p1382] the "in body[p1432]" insertion mode[p1382].

↪ **A comment token**

Insert a comment[p1423] given (the first element in the stack of open elements[p1383] (the `html`[p179] element), null).

↪ **A processing instruction token**

Insert a processing instruction[p1423] given (the first element in the stack of open elements[p1383] (the `html`[p179] element), null).

↪ **A DOCTYPE token**

Parse error[p1370]. Ignore the token.

↪ **A start tag whose tag name is "html"**

Process the token using the rules for[p1382] the "in body[p1432]" insertion mode[p1382].

↪ **An end tag whose tag name is "html"**

If the parser's fragment context element[p1386] is non-null, this is a parse error[p1370]; ignore the token. (fragment case[p1473])

Otherwise, switch the insertion mode[p1382] to "after after body[p1455]".

↪ **An end-of-file token**

Stop parsing[p1458].

↪ **Anything else**

Parse error[p1370]. Switch the insertion mode[p1382] to "in body[p1432]" and reprocess the token.

13.2.6.4.18 The "in frameset" insertion mode §[p14 53]

When the user agent is to apply the rules for the "in frameset[p1453]" insertion mode[p1382], 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[p1422].

↪ **A comment token**

Insert a comment[p1423].

↪ **A processing instruction token**

Insert a processing instruction[p1423].

↪ **A DOCTYPE token**

Parse error[p1370]. Ignore the token.

↪ **A start tag whose tag name is "html"**

Process the token using the rules for[p1382] the "in body[p1432]" insertion mode[p1382].

↪ **A start tag whose tag name is "frameset"**

Insert an HTML element[p1420] for the token.

↪ **An end tag whose tag name is "frameset"**

If the current node[p1384] is the root `html`[p179] element, then this is a parse error[p1370]; ignore the token. (fragment case[p1473])

Otherwise, pop the current node[p1384] from the stack of open elements[p1383].

If the parser's fragment context element[p1386] is null and the current node[p1384] is no longer a `frameset`[p1537] element, then switch the insertion mode[p1382] to "after frameset[p1454]".

↪ **A start tag whose tag name is "frame"**

Insert an HTML element[p1420] for the token. Immediately pop the current node[p1384] off the stack of open elements[p1383].

Acknowledge the token's *self-closing flag*[p1387], if it is set.

↪ **A start tag whose tag name is "noframes"**

Process the token using the rules for[p1382] the "in head[p1427]" insertion mode[p1382].

↪ **An end-of-file token**

If the current node[p1384] is not the root `html`[p179] element, then this is a parse error[p1370].

Note

The current node[p1384] can only be the root `html`[p179] element in the fragment case[p1473].

Stop parsing[p1458].

↪ **Anything else**

Parse error[p1370]. Ignore the token.

13.2.6.4.19 The "after frameset" insertion mode §[p1454]

When the user agent is to apply the rules for the "after frameset[p1454]" insertion mode[p1382], 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[p1422].

↪ **A comment token**

Insert a comment[p1423].

↪ **A processing instruction token**

Insert a processing instruction[p1423].

↪ **A DOCTYPE token**

Parse error[p1370]. Ignore the token.

↪ **A start tag whose tag name is "html"**

Process the token using the rules for[p1382] the "in body[p1432]" insertion mode[p1382].

↪ **An end tag whose tag name is "html"**

Switch the insertion mode[p1382] to "after after frameset[p1455]".

↪ **A start tag whose tag name is "noframes"**

Process the token using the rules for[p1382] the "in head[p1427]" insertion mode[p1382].

↪ **An end-of-file token**

Stop parsing[p1458].

↪ **Anything else**

Parse error[p1370]. Ignore the token.

13.2.6.4.20 The "after after body" insertion mode §[p14 55]

When the user agent is to apply the rules for the "after after body[p1455]" insertion mode[p1382], the user agent must handle the token as follows:

↪ **A comment token**

Insert a comment[p1423] given (the Document[p135] object, null).

↪ **A processing instruction token**

Insert a processing instruction[p1423] given (the Document[p135] object, null).

↪ **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[p1382] the "in body[p1432]" insertion mode[p1382].

↪ **An end-of-file token**

Stop parsing[p1458].

↪ **Anything else**

Parse error[p1370]. Switch the insertion mode[p1382] to "in body[p1432]" and reprocess the token.

13.2.6.4.21 The "after after frameset" insertion mode §[p14 55]

When the user agent is to apply the rules for the "after after frameset[p1455]" insertion mode[p1382], the user agent must handle the token as follows:

↪ **A comment token**

Insert a comment[p1423] given (the Document[p135] object, null).

↪ **A processing instruction token**

Insert a processing instruction[p1423] given (the Document[p135] object, null).

↪ **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[p1382] the "in body[p1432]" insertion mode[p1382].

↪ **An end-of-file token**

Stop parsing[p1458].

↪ **A start tag whose tag name is "noframes"**

Process the token using the rules for[p1382] the "in head[p1427]" insertion mode[p1382].

↪ **Anything else**

Parse error[p1370]. Ignore the token.

13.2.6.5 The rules for parsing tokens in foreign content §[p14 55]

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[p1370]. Insert a U+FFFD REPLACEMENT CHARACTER character[p1422].

↪ **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[p1422].

↪ **Any other character token**

Insert the token's character[p1422].

Set the frameset-ok flag[p1387] to "not ok".

↪ **A comment token**

Insert a comment[p1423].

↪ **A processing instruction token**

Insert a processing instruction[p1423].

↪ **A DOCTYPE token**

Parse error[p1370]. Ignore the token.

↪ **A start tag whose tag name is one of: "b", "big", "blockquote", "body", "br", "center", "code", "dd", "div", "dl", "dt", "em", "embed", "h1", "h2", "h3", "h4", "h5", "h6", "head", "hr", "i", "img", "li", "listing", "menu", "meta", "nobr", "ol", "p", "pre", "ruby", "s", "small", "span", "strong", "strike", "sub", "sup", "table", "tt", "u", "ul", "var"**

↪ **A start tag whose tag name is "font", if the token has any attributes named "color", "face", or "size"**

↪ **An end tag whose tag name is "br", "p"**

Parse error[p1370].

While the current node[p1384] is not a MathML text integration point[p1417], an HTML integration point[p1417], or an element in the HTML namespace, pop elements from the stack of open elements[p1383].

Reprocess the token according to the rules given in the section corresponding to the current insertion mode[p1382] in HTML content.

↪ **Any other start tag**

If the adjusted current node[p1384] is an element in the MathML namespace, adjust MathML attributes[p1420] for the token. (This fixes the case of MathML attributes that are not all lowercase.)

If the adjusted current node[p1384] is an element in the SVG namespace, and the token's tag name is one of the ones in the first column of the following table, change the tag name to the name given in the corresponding cell in the second column. (This fixes the case of SVG elements that are not all lowercase.)

Tag name	Element name
altglyph	altGlyph
altglyphdef	altGlyphDef
altglyphitem	altGlyphItem
animatecolor	animateColor
animatemotion	animateMotion
animatetransform	animateTransform
clippath	clipPath
feblend	feBlend
fecolormatrix	feColorMatrix
fecomponenttransfer	feComponentTransfer
fecomposite	feComposite
feconvolvematrix	feConvolveMatrix
fediffuselighting	feDiffuseLighting
fedisplacementmap	feDisplacementMap
fedistantlight	feDistantLight
fedropshadow	feDropShadow
feflood	feFlood
fefunca	feFuncA

Tag name	Element name
fefuncb	feFuncB
fefuncg	feFuncG
fefuncr	feFuncR
fegaussianblur	feGaussianBlur
feimage	feImage
femerge	feMerge
femergenode	feMergeNode
femorphology	feMorphology
feoffset	feOffset
fepointlight	fePointLight
fespecularlighting	feSpecularLighting
fespotlight	feSpotLight
fetile	feTile
feturbulence	feTurbulence
foreignobject	foreignObject
glyphref	glyphRef
lineargradient	linearGradient
radialgradient	radialGradient
textpath	textPath

If the adjusted current node[p1384] is an element in the SVG namespace, adjust SVG attributes[p1420] for the token. (This fixes the case of SVG attributes that are not all lowercase.)

Adjust foreign attributes[p1421] for the token. (This fixes the use of namespaced attributes, in particular XLink in SVG.)

Insert a foreign element[p1420] for the token, with the adjusted current node[p1384]'s namespace and false.

If the token has its *self-closing flag*[p1387] set, then run the appropriate steps from the following list:

↪ **If the token's tag name is "script", and the new current node[p1384] is in the SVG namespace**
Acknowledge the token's *self-closing flag*[p1387], and then act as described in the steps for a "script" end tag below.

↪ **Otherwise**
Pop the current node[p1384] off the stack of open elements[p1383] and acknowledge the token's *self-closing flag*[p1387].

↪ **An end tag whose tag name is "script", if the current node[p1384] is an SVG `script` element**

Pop the current node[p1384] off the stack of open elements[p1383].

Let the *old insertion point* have the same value as the current insertion point[p1382]. Let the insertion point[p1382] be just before the next input character[p1382].

Increment the parser's script nesting level[p1370] by one. Set the parser pause flag[p1370] to true.

If the active speculative HTML parser[p1459] is null and the user agent supports SVG, then Process the SVG `script` element according to the SVG rules. [SVG][p1586]

Note

Even if this causes new characters to be inserted into the tokenizer[p1223], the parser will not be executed reentrantly, since the parser pause flag[p1370] is true.

Decrement the parser's script nesting level[p1370] by one. If the parser's script nesting level[p1370] is zero, then set the parser pause flag[p1370] to false.

Let the insertion point[p1382] have the value of the *old insertion point*. (In other words, restore the insertion point[p1382] 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[p1384] (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[p1370].

3. *Loop*: If *node* is the topmost element in the stack of open elements[p1383], then return. (fragment case[p1473])

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[p1383] until *node* has been popped from the stack, and then return.

5. Set *node* to the previous entry in the stack of open elements[p1383].

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[p1382] in HTML content.

13.2.7 The end §[p1458]

Once the user agent **stops parsing** the document, the user agent must run the following steps:

✔ MDN

1. If the active speculative HTML parser[p1459] is not null, then stop the speculative HTML parser[p1460] and return.

2. Set the insertion point[p1382] to undefined.

3. Update the current document readiness[p141] to "`interactive`".

4. Pop *all* the nodes off the stack of open elements[p1383].

5. While the list of scripts that will execute when the document has finished parsing[p698] is not empty:

    1. Spin the event loop[p1201] until the first `script`[p685] in the list of scripts that will execute when the document has finished parsing[p698] has its ready to be parser-executed[p692] set to true *and* the parser's `Document`[p135] has no style sheet that is blocking scripts[p212].

    2. Execute the script element[p698] given by the first `script`[p685] in the list of scripts that will execute when the document has finished parsing[p698].

    3. Remove the first `script`[p685] element from the list of scripts that will execute when the document has finished parsing[p698] (i.e. shift out the first entry in the list).

6. Queue a global task[p1195] on the DOM manipulation task source[p1203] given the `Document`[p135]'s relevant global object[p1151] to run the following substeps:

    1. Set the `Document`[p135]'s load timing info[p141]'s DOM content loaded event start time[p142] to the current high resolution time given the `Document`[p135]'s relevant global object[p1151].

    2. Fire an event named `DOMContentLoaded`[p1574] at the `Document`[p135] object, with its `bubbles` attribute initialized to true.

    3. Set the `Document`[p135]'s load timing info[p141]'s DOM content loaded event end time[p142] to the current high resolution time given the `Document`[p135]'s relevant global object[p1151].

    4. Enable the client message queue of the `ServiceWorkerContainer` object whose associated service worker client is the `Document`[p135] object's relevant settings object[p1151].

    5. Invoke WebDriver BiDi DOM content loaded with the `Document`[p135]'s browsing context[p1046], and a new WebDriver BiDi navigation status whose id is the `Document`[p135] object's during-loading navigation ID for WebDriver BiDi[p136], status is "`pending`", and url is the `Document`[p135] object's URL.

7. Spin the event loop[p1201] until the set of scripts that will execute as soon as possible[p698] and the list of scripts that will execute in order as soon as possible[p698] are empty.

8. Spin the event loop[p1201] until there is nothing that **delays the load event** in the `Document`[p135].

9. Queue a global task[p1195] on the DOM manipulation task source[p1203] given the `Document`[p135]'s relevant global object[p1151] to run the following steps:

    1. Update the current document readiness[p141] to "`complete`".

2. If the Document object's browsing context is null, then abort these steps.
3. Let *window* be the Document's relevant global object.
4. Set the Document's load timing info's load event start time to the current high resolution time given *window*.
5. Fire an event named load at *window*, with *legacy target override flag* set.
6. Invoke WebDriver BiDi load complete with the Document's browsing context, and a new WebDriver BiDi navigation status whose id is the Document object's during-loading navigation ID for WebDriver BiDi, status is "complete", and url is the Document object's URL.
7. Set the Document object's during-loading navigation ID for WebDriver BiDi to null.
8. Set the Document's load timing info's load event end time to the current high resolution time given *window*.
9. Assert: Document's page showing is false.
10. Set the Document's page showing to true.
11. Fire a page transition event named pageshow at *window* with false.
12. Completely finish loading the Document.
13. Queue the navigation timing entry for the Document.

10. If the Document's print when loaded flag is set, then run the printing steps.
11. The Document is now **ready for post-load tasks**.

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. Stop the speculative HTML parser for this HTML parser.
3. Update the current document readiness to "interactive".
4. Pop *all* the nodes off the stack of open elements.
5. Update the current document readiness to "complete".

13.2.8 Speculative HTML parsing

User agents may implement an optimization, as described in this section, to speculatively fetch resources that are declared in the HTML markup while the HTML parser is waiting for a pending parsing-blocking script to be fetched and executed, or during normal parsing, at the time an element is created for a token. While this optimization is not defined in precise detail, there are some rules to consider for interoperability.

Each HTML parser can have an **active speculative HTML parser**. It is initially null.

The **speculative HTML parser** must act like the normal HTML parser (e.g., the tree builder rules apply), with some exceptions:

- The state of the normal HTML parser and the document itself must not be affected.

  Example

  For example, the next input character or the stack of open elements for the normal HTML parser is not affected by the speculative HTML parser.

- Bytes pushed into the HTML parser's input byte stream must also be pushed into the speculative HTML parser's input byte stream. Bytes read from the streams must be independent.
- The result of the speculative parsing is primarily a series of speculative fetches. Which kinds of resources to speculatively fetch is implementation-defined, but user agents must not speculatively fetch resources that would not be

fetched with the normal HTML parser, under the assumption that the script that is blocking the HTML parser does nothing.

Note

It is possible that the same markup is seen multiple times from the speculative HTML parser[p1459] and then the normal HTML parser. It is expected that duplicated fetches will be prevented by caching rules, which are not yet fully specified.

A **speculative fetch** for a speculative mock element[p1460] *element* must follow these rules:

Should some of these things be applied to the document "for real", even though they are found speculatively?

- If the speculative HTML parser[p1459] encounters one of the following elements, then act as if that element is processed for the purpose of its effect of subsequent speculative fetches.
    - A `base`[p182] element.
    - A `meta`[p196] element whose `http-equiv`[p202] attribute is in the Content security policy[p206] state.
    - A `meta`[p196] element whose `name`[p197] attribute is an ASCII case-insensitive match for "`referrer`[p199]".
    - A `meta`[p196] element whose `name`[p197] attribute is an ASCII case-insensitive match for "`viewport`". (This can affect whether a media query list matches the environment[p100].) [CSSDEVICEADAPT][p1580]
- Let *url* be the URL that *element* would fetch if it was processed normally. If there is no such URL or if it is the empty string, then do nothing. Otherwise, if *url* is already in the list of speculative fetch URLs[p1460], then do nothing. Otherwise, fetch *url* as if the element was processed normally, and add *url* to the list of speculative fetch URLs[p1460].

Each `Document`[p135] has a **list of speculative fetch URLs**, which is a list of URLs, initially empty.

To **start the speculative HTML parser** for an instance of an HTML parser *parser*:

1. Optionally, return.

    Note

    This step allows user agents to opt out of speculative HTML parsing.

2. If *parser*'s active speculative HTML parser[p1459] is not null, then stop the speculative HTML parser[p1460] for *parser*.

    Note

    This can happen when `document.write()`[p1223] writes another parser-blocking script. For simplicity, this specification always restarts speculative parsing, but user agents can implement a more efficient strategy, so long as the end result is equivalent.

3. Let *speculativeParser* be a new speculative HTML parser[p1459], with the same state as *parser*.
4. Let *speculativeDoc* be a new isomorphic representation of *parser*'s `Document`[p135], where all elements are instead speculative mock elements[p1460]. Let *speculativeParser* parse into *speculativeDoc*.
5. Set *parser*'s active speculative HTML parser[p1459] to *speculativeParser*.
6. In parallel[p44], run *speculativeParser* until it is stopped or until it reaches the end of its input stream[p1382].

To **stop the speculative HTML parser** for an instance of an HTML parser *parser*:

1. Let *speculativeParser* be *parser*'s active speculative HTML parser[p1459].
2. If *speculativeParser* is null, then return.
3. Throw away any pending content in *speculativeParser*'s input stream[p1382], and discard any future content that would have been added to it.
4. Set *parser*'s active speculative HTML parser[p1459] to null.

The speculative HTML parser[p1459] will create speculative mock elements[p1460] instead of normal elements. DOM operations that the tree builder normally does on elements are expected to work appropriately on speculative mock elements.

A **speculative mock element** is a struct with the following items:

- A string **namespace**, corresponding to an element's namespace.

- A string **local name**, corresponding to an element's local name.
- A list **attribute list**, corresponding to an element's attribute list.
- A list **children**, corresponding to an element's children.

To **create a speculative mock element** given a *namespace*, *tagName*, and *attributes*:

1. Let *element* be a new speculative mock element[p1460].
2. Set *element*'s namespace[p1460] to *namespace*.
3. Set *element*'s local name[p1461] to *tagName*.
4. Set *element*'s attribute list[p1461] to *attributes*.
5. Set *element*'s children[p1461] to a new empty list.
6. Optionally, perform a speculative fetch[p1460] for *element*.
7. Return *element*.

When the tree builder says to insert an element into a `template`[p705] element's template contents[p707], if that is a speculative mock element[p1460], and the `template`[p705] element's template contents[p707] is not a `ShadowRoot` node, instead do nothing. URLs found speculatively inside non-declarative-shadow-root `template`[p705] elements might themselves be templates, and must not be speculatively fetched.

13.2.9 Coercing an HTML DOM into an infoset §[p1461]

When an application uses an HTML parser[p1368] in conjunction with an XML pipeline, it is possible that the constructed DOM is not compatible with the XML tool chain in certain subtle ways. For example, an XML toolchain might not be able to represent attributes with the name `xmlns`, since they conflict with the Namespaces in XML syntax. There is also some data that the HTML parser[p1368] 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.

Example

For example, the element name `foo<bar`, which can be output by the HTML parser[p1368], though it is neither a legal HTML element name nor a well-formed XML element name, would be converted into `fooU00003Cbar`, which *is* a well-formed XML element name (though it's still not legal in HTML by any means).

Example

As another example, consider the attribute `xlink:href`. Used on a MathML element, it becomes, after being adjusted[p1421], an attribute with a prefix "`xlink`" and a local name "`href`". However, used on an HTML element, it becomes an attribute with no prefix and the local name "`xlink:href`", which is not a valid NCName, and thus might not be accepted by an XML API. It could thus get converted, becoming "`xlinkU00003Ahref`".

Note

The resulting names from this conversion conveniently can't clash with any attribute generated by the HTML parser[p1368], since those are all either lowercase or those listed in the adjust foreign attributes[p1421] algorithm's table.

If the XML API restricts comments from having two consecutive U+002D HYPHEN-MINUS characters (--), the tool may insert a single U+0020 SPACE character between any such offending characters.

If the XML API restricts comments from ending in a U+002D HYPHEN-MINUS character (-), the tool may insert a single U+0020 SPACE character at the end of such comments.

If the XML API restricts allowed characters in character data, attribute values, or comments, the tool may replace any U+000C FORM FEED (FF) character with a U+0020 SPACE character, and any other literal non-XML character with a U+FFFD REPLACEMENT CHARACTER.

If the tool has no way to convey out-of-band information, then the tool may drop the following information:

- Whether the document is set to *no-quirks mode*, *limited-quirks mode*, or *quirks mode*
- The association between form controls and forms that aren't their nearest `form` element ancestor (use of the form element pointer in the parser)
- The template contents of any `template` elements.

Note

The mutations allowed by this section apply after the HTML parser's rules have been applied. For example, a `<a::>` start tag will be closed by a `</a::>` end tag, and never by a `</aU00003AU00003A>` end tag, even if the user agent is using the rules above to then generate an actual element in the DOM with the name `aU00003AU00003A` for that start tag.

13.2.10 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.

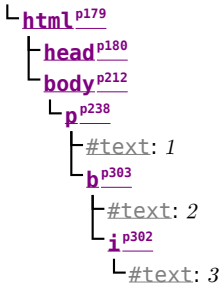
13.2.10.1 Misnested tags:

This section is non-normative.

The most-often discussed example of erroneous markup is as follows:

```
<p>1<b>2<i>3</b>4</i>5</p>
```

The parsing of this markup is straightforward up to the "3". At this point, the DOM looks like this:

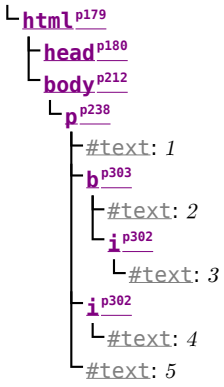


Here, the stack of open elements has five elements on it: `html`, `body`, `p`, `b`, and `i`. The list of active formatting elements just has two: `b` and `i`. The insertion mode is "in body".

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 *formattingElement* 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:



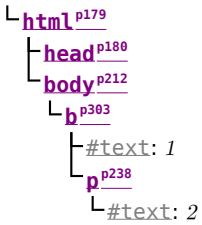
13.2.10.2 Misnested tags:

This section is non-normative.

A case similar to the previous one is the following:

```
<b>1<p>2</b>3</p>
```

Up to the "2" the parsing here is straightforward:



The interesting part is when the end tag token with the tag name "b" is parsed.

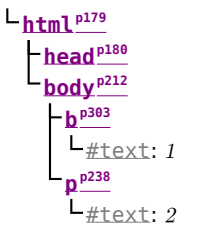
Before that token is seen, the stack of open elements has four elements on it: html, body, b, and p. The list of active formatting elements just has the one: b. The insertion mode is "in body".

Upon receiving the end tag token with the tag name "b", the "adoption agency algorithm" is invoked, as in the previous example. However, in this case, there *is* a *furthest block*, namely the p element. Thus, this time the adoption agency algorithm isn't skipped over.

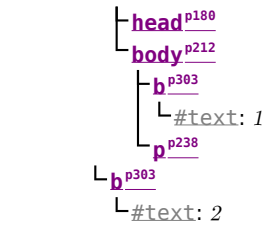
The *common ancestor* is the body element. A conceptual "bookmark" marks the position of the b in the list of active formatting elements, but since that list has only one element in it, the bookmark won't have much effect.

As the algorithm progresses, *node* ends up set to the formatting element (b), and *last node* ends up set to the *furthest block* (p).

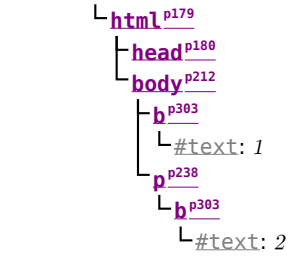
The *last node* gets appended (moved) to the *common ancestor*, so that the DOM looks like:



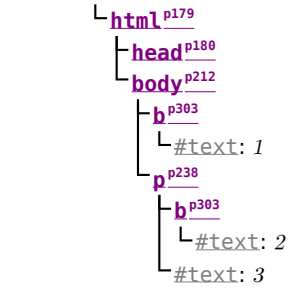
A new b element is created, and the children of the p element are moved to it:



Finally, the new b[p303] element is appended to the p[p238] element, so that the DOM looks like:



The b[p303] element is removed from the list of active formatting elements[p1385] and the stack of open elements[p1383], so that when the "3" is parsed, it is appended to the p[p238] element:



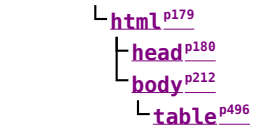
13.2.10.3 Unexpected markup in tables §[p1464]

This section is non-normative.

Error handling in tables is, for historical reasons, especially strange. For example, consider the following markup:

```
<table><b><tr><td>aaa</td></tr>bbb</table>ccc
```

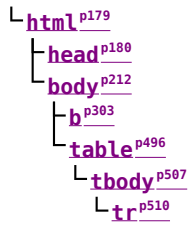
The highlighted b[p303] element start tag is not allowed directly inside a table like that, and the parser handles this case by placing the element *before* the table. (This is called *foster parenting*[p1418].) This can be seen by examining the DOM tree as it stands just after the table[p496] element's start tag has been seen:



...and then immediately after the b[p303] element start tag has been seen:

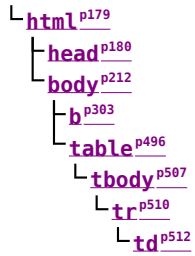
At this point, the stack of open elements[p1383] has on it the elements html[p179], body[p212], table[p496], and b[p303] (in that order, despite the resulting DOM tree); the list of active formatting elements[p1385] just has the b[p303] element in it; and the insertion mode[p1382] is "in table[p1445]".

The tr[p510] start tag causes the b[p303] element to be popped off the stack and a tbody[p507] start tag to be implied; the tbody[p507] and tr[p510] elements are then handled in a rather straight-forward manner, taking the parser through the "in table body[p1449]" and "in row[p1450]" insertion modes, after which the DOM looks as follows:

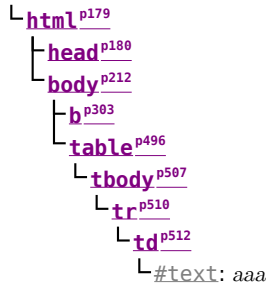


Here, the stack of open elements[p1383] has on it the elements html[p179], body[p212], table[p496], tbody[p507], and tr[p510]; the list of active formatting elements[p1385] still has the b[p303] element in it; and the insertion mode[p1382] is "in row[p1450]".

The td[p512] element start tag token, after putting a td[p512] element on the tree, puts a marker[p1385] on the list of active formatting elements[p1385] (it also switches to the "in cell[p1451]" insertion mode[p1382]).



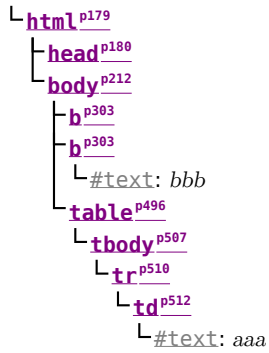
The marker[p1385] means that when the "aaa" character tokens are seen, no b[p303] element is created to hold the resulting Text node:



The end tags are handled in a straight-forward manner; after handling them, the stack of open elements[p1383] has on it the elements html[p179], body[p212], table[p496], and tbody[p507]; the list of active formatting elements[p1385] still has the b[p303] element in it (the marker[p1385] having been removed by the "td" end tag token); and the insertion mode[p1382] is "in table body[p1449]".

Thus it is that the "bbb" character tokens are found. These trigger the "in table text[p1447]" insertion mode to be used (with the original insertion mode[p1382] set to "in table body[p1449]"). The character tokens are collected, and when the next token (the table[p496] element end tag) is seen, they are processed as a group. Since they are not all spaces, they are handled as per the "anything else" rules in the "in table[p1445]" insertion mode, which defer to the "in body[p1432]" insertion mode but with foster parenting[p1418].

When the active formatting elements are reconstructed[p1386], a b[p303] element is created and foster parented[p1418], and then the "bbb" Text node is appended to it:



The stack of open elements[p1383] has on it the elements html[p179], body[p212], table[p496], tbody[p507], and the new b[p303] (again, note that this doesn't match the resulting tree!); the list of active formatting elements[p1385] has the new b[p303] element in it; and the insertion mode[p1382] is still "in table body[p1449]".

Had the character tokens been only ASCII whitespace instead of "bbb", then that ASCII whitespace would just be appended to the tbody[p507] element.

Finally, the table[p496] is closed by a "table" end tag. This pops all the nodes from the stack of open elements[p1383] up to and including the table[p496] element, but it doesn't affect the list of active formatting elements[p1385], so the "ccc" character tokens after the table result in yet another b[p303] element being created, this time after the table:

- html[p179]
  - head[p180]
  - body[p212]
    - b[p303]
    - b[p303]
      - #text: *bbb*
    - table[p496]
      - tbody[p507]
        - tr[p510]
          - td[p512]
            - #text: *aaa*
    - b[p303]
      - #text: *ccc*

13.2.10.4 Scripts that modify the page as it is being parsed §[p1466]

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[p404] in another document with the URL `https://example.com/outer`:

```
<div id=a>
 <script>
  var div = document.getElementById('a');
  parent.document.body.appendChild(div);
 </script>
 <script>
  alert(document.URL);
 </script>
</div>
<script>
 alert(document.URL);
</script>
```

Up to the first "script" end tag, before the script is parsed, the result is relatively straightforward:

- html[p179]
  - head[p180]
  - body[p212]
    - div[p266] id[p162]="a"
      - #text:
      - script[p685]
        - #text: *var div = document.getElementById('a'); parent.document.body.appendChild(div);*

After the script is parsed, though, the div[p266] element and its child script[p685] element are gone:

- html[p179]
  - head[p180]
  - body[p212]

They are, at this point, in the Document[p135] of the aforementioned outer browsing context[p1046]. However, the stack of open elements[p1383] *still contains the* div[p266] *element*.

Thus, when the second script[p685] element is parsed, it is inserted *into the outer* Document[p135] *object*.

Those parsed into different Document[p135]s than the one the parser was created for do not execute, so the first alert does not show.

Once the div[p266] element's end tag is parsed, the div[p266] element is popped off the stack, and so the next script[p685] element is in the inner Document[p135]:

```
└html[p179]
  ├head[p180]
  └body[p212]
    └script[p685]
      └#text: alert(document.URL);
```

This script does execute, resulting in an alert that says "https://example.com/inner".

13.2.10.5 The execution of scripts that are moving across multiple documents §[p14 67]

This section is non-normative.

Elaborating on the example in the previous section, consider the case where the second script[p685] element is an external script (i.e. one with a src[p686] attribute). Since the element was not in the parser's Document[p135] when it was created, that external script is not even downloaded.

In a case where a script[p685] element with a src[p686] attribute is parsed normally into its parser's Document[p135], 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[p685] *elements between* Document[p135]*s is considered a bad practice.*

13.2.10.6 Unclosed formatting elements §[p14 67]

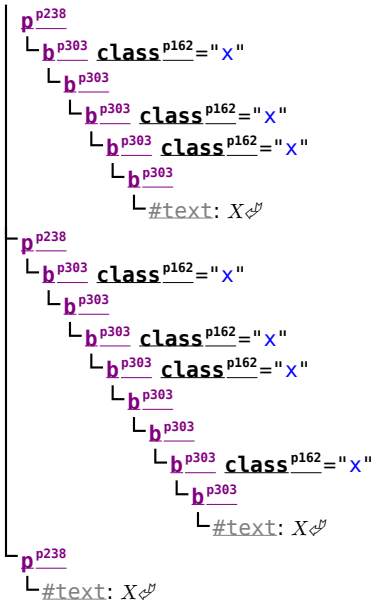
This section is non-normative.

The following markup shows how nested formatting elements (such as b[p303]) get collected and continue to be applied even as the elements they are contained in are closed, but that excessive duplicates are thrown away.

```
<!DOCTYPE html>
<p><b class=x><b class=x><b><b class=x><b class=x><b>X
<p>X
<p><b><b class=x><b>X
<p></b></b></b></b></b></b>X
```

The resulting DOM tree is as follows:

```
├DOCTYPE: html
└html[p179]
  ├head[p180]
  └body[p212]
    ├p[p238]
    │ └b[p303] class[p162]="x"
    │   └b[p303] class[p162]="x"
    │     └b[p303]
    │       └b[p303] class[p162]="x"
    │         └b[p303] class[p162]="x"
    │           └b[p303]
    │             └#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`, `bgsound`, `frame`, `keygen`, or `param`.

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*, a boolean *serializableShadowRoots*, and a `sequence<ShadowRoot>` *shadowRoots*, 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 `DocumentFragment` node).
4. If *current node* is a shadow host:
    1. Let *shadow* be *current node*'s shadow root.
    2. If one of the following is true:
        - *serializableShadowRoots* is true and *shadow*'s serializable is true; or
        - *shadowRoots* contains *shadow*,

        then:
        1. Append "`<template shadowrootmode="`".
        2. If *shadow*'s mode is "`open`", then append "`open`". Otherwise, append "`closed`".

3. Append U+0022 (").
4. If *shadow*'s delegates focus is set, then append " `shadowrootdelegatesfocus=""`".
5. If *shadow*'s serializable is set, then append " `shadowrootserializable=""`".
6. If *shadow*'s slot assignment is "`manual`", then append " `shadowrootslotassignment="manual"`".
7. If *shadow*'s clonable is set, then append " `shadowrootclonable=""`".
8. Let *shouldAppendRegistryAttribute* be the result of running these steps:
    1. Let *documentRegistry* be *shadow*'s node document's custom element registry.
    2. Let *shadowRegistry* be *shadow*'s custom element registry.
    3. If *documentRegistry* is null and *shadowRegistry* is null, then return false.
    4. If *documentRegistry* is a global custom element registry and *shadowRegistry* is a global custom element registry, then return false.
    5. Return true.
9. If *shouldAppendRegistryAttribute* is true, then append " `shadowrootcustomelementregistry=""`".
10. Append U+003E (>).
11. Append the value of running the HTML fragment serialization algorithm[p1468] with *shadow*, *serializableShadowRoots*, and *shadowRoots* (thus recursing into this algorithm for that element).
12. Append "`</template>`".

5. 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 U+003C (<), followed by *tagname*.

        Note

        For HTML elements[p46] created by the HTML parser[p1368] or `createElement()`, tagname will be lowercase.

        If *current node*'s `is` value is not null, and the element does not have an `is`[p796] attribute in its attribute list, then append " `is=""`", followed by *current node*'s `is` value escaped as described below[p1472] in *attribute mode*, and U+0022 (").

        For each attribute that the element has, append U+0020 SPACE, the attribute's serialized name as described below[p1469], "`=""`", the attribute's value, escaped as described below[p1472] in *attribute mode*, and U+0022 (").

        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[p46] set by the HTML parser[p1368] or by `setAttribute()`, the local name will be lowercase.

        ↪ **If the attribute is in the XML namespace**
        The attribute's serialized name is "`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 "`xmlns`".

↪ **If the attribute is in the XMLNS namespace and the attribute's local name is not `xmlns`**
The attribute's serialized name is "`xmlns:`" followed by the attribute's local name.

↪ **If the attribute is in the XLink namespace**
The attribute's serialized name is "`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 U+003E (>).

If *current node* serializes as void[p1468], then continue on to the next child node at this point.

Append the value of running the HTML fragment serialization algorithm[p1468] with *current node*, *serializableShadowRoots*, and *shadowRoots* (thus recursing into this algorithm for that node), followed by "</", *tagname*, and U+003E (>).

↪ **If *current node* is a `Text` node**
If the parent of *current node* is a `style`[p207], `script`[p685], `xmp`[p1531], `iframe`[p404], `noembed`[p1530], `noframes`[p1530], or `plaintext`[p1530] element, or if the parent of *current node* is a `noscript`[p703] element and scripting is enabled[p1152] for the node, then append the value of *current node*'s data literally.

Otherwise, append the value of *current node*'s data, escaped as described below[p1472].

↪ **If *current node* is a `Comment`**
Append "<!--", followed by the value of *current node*'s data, and "-->".

↪ **If *current node* is a `ProcessingInstruction`**
Append "<?", followed by the value of *current node*'s target, U+0020 SPACE, the value of *current node*'s data, and "?>".

↪ **If *current node* is a `DocumentType`**
Append "<!DOCTYPE ", followed by the value of *current node*'s name and U+003E (>).

6. Return *s*.

⚠Warning!

It is possible that the output of this algorithm, if parsed with an HTML parser[p1368], 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[p1368] itself, although such cases are typically non-conforming.

¶[p1470] Example

For instance, if a `textarea`[p606] 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 "-->", 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`[p685] element contain a `Text` node with the text string "</script>", or having a `p`[p238] element that contains a `ul`[p249] element (as the `ul`[p249] element's start tag[p1358] would imply the end tag for the `p`[p238]).

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`[p207] block via the DOM and which then uses the `innerHTML`[p1229] IDL attribute to get the HTML serialization of that `style`[p207] element: if the user enters "</style><script>attack</script>" as a font family name, `innerHTML`[p1229] will return markup that, if parsed in a different context, would contain a `script`[p685] node, even though no `script`[p685] node existed in the original DOM.

¶ p14 71 **Example**

For example, consider the following markup:

```
<form id="outer"><div></form><form id="inner"><input>
```

This will be parsed into:

```
└ html p179
  ├ head p180
  └ body p212
    └ form p532 id p162 ="outer"
      └ div p266
        └ form p532 id p162 ="inner"
          └ input p539
```

The input p539 element will be associated with the inner form p532 element. Now, if this tree structure is serialized and reparsed, the <form id="inner"> start tag will be ignored, and so the input p539 element will be associated with the outer form p532 element instead.

```
<html><head></head><body><form id="outer"><div><form
id="inner"><input></form></div></form></body></html>
```

```
└ html p179
  ├ head p180
  └ body p212
    └ form p532 id p162 ="outer"
      └ div p266
        └ input p539
```

¶ p14 71 **Example**

As another example, consider the following markup:

```
<a><table><a>
```

This will be parsed into:

```
└ html p179
  ├ head p180
  └ body p212
    └ a p267
      ├ a p267
      └ table p496
```

That is, the a p267 elements are nested, because the second a p267 element is foster parented p1418. After a serialize-reparse roundtrip, the a p267 elements and the table p496 element would all be siblings, because the second <a> start tag implicitly closes the first a p267 element.

```
<html><head></head><body><a><a></a><table></table></a></body></html>
```

```
└ html p179
  ├ head p180
  └ body p212
    ├ a p267
    ├ a p267
    └ table p496
```

For historical reasons, this algorithm does not roundtrip an initial U+000A (LF) character in pre p242, textarea p606, or listing p1530 elements, even though (in the first two cases) the markup being roundtripped can be conforming. The HTML parser p1368 will drop such a character during parsing, but this algorithm does *not* serialize an extra U+000A (LF) character.

¶ p14 72 **Example**

For example, consider the following markup:

```
<pre>

Hello.</pre>
```

When this document is first parsed, the pre p242 element's child text content starts with a single newline character. After a serialize-reparse roundtrip, the pre p242 element's child text content is simply "Hello.".

Because of the special role of the is p796 attribute in signaling the creation of customized built-in elements p796, 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.

¶ p14 72 **Example**

When creating a customized built-in element p796 via the parser, a developer uses the is p796 attribute directly; in such cases serialize-parse roundtrips work fine.

```
<script>
window.SuperP = class extends HTMLParagraphElement {};
customElements.define("super-p", SuperP, { extends: "p" });
</script>

<div id="container"><p is="super-p">Superb!</p></div>

<script>
console.log(container.innerHTML); // <p is="super-p">
container.innerHTML = container.innerHTML;
console.log(container.innerHTML); // <p is="super-p">
console.assert(container.firstChild instanceof SuperP);
</script>
```

But when creating a customized built-in element via its constructor p796 or via createElement(), the is p796 attribute is not added. Instead, the is value (which is what the custom elements machinery uses) is set without intermediating through an attribute.

```
<script>
container.innerHTML = "";
const p = document.createElement("p", { is: "super-p" });
container.appendChild(p);

// The is attribute is not present in the DOM:
console.assert(!p.hasAttribute("is"));

// But the element is still a super-p:
console.assert(p instanceof SuperP);
</script>
```

To ensure that serialize-parse roundtrips still work, the serialization process explicitly writes out the element's is value as an is p796 attribute:

```
<script>
console.log(container.innerHTML); // <p is="super-p">
container.innerHTML = container.innerHTML;
console.log(container.innerHTML); // <p is="super-p">
console.assert(container.firstChild instanceof SuperP);
</script>
```

Escaping a string (for the purposes of the algorithm above) consists of running the following steps:

1. Replace any occurrence of "&" character by "&".

2. Replace any occurrences of the U+00A0 NO-BREAK SPACE character by "` `".
3. Replace any occurrences of the "<" character by "`<`".
4. Replace any occurrences of the ">" character by "`>`".
5. If the algorithm was invoked in the *attribute mode*, then replace any occurrences of the "`"`" character by "`"`".

13.4 Parsing HTML fragments §[p1473]

The **HTML fragment parsing algorithm**, given an `Element` or `DocumentFragment` *target*, a string *input*, an optional boolean *allowDeclarativeShadowRoots* (default false), and an optional parser scripting mode[p1387] *scriptingMode* (default Inert[p1387]), is the following steps. They return a `DocumentFragment`.

Note

Parts marked **fragment case** *in algorithms in the HTML parser[p1368] section are parts that only occur if the parser's fragment context element[p1386] is non-null. The algorithms have been annotated with such markings for informational purposes only; such markings have no normative weight. If it is possible for a condition described as a fragment case[p1473] to occur even when the parser's fragment context element[p1386] is null, then that is an error in the specification.*

1. Assert: *scriptingMode* is either Inert[p1387] or Fragment[p1387].
2. Let ***context*** be *target* if *target* is an `Element`; otherwise *target*'s host.
3. Assert: *context*[p1473] is non-null.
4. Let *document* be a `Document`[p135] node whose type is "`html`".
5. Let *contextDocument* be *context*[p1473]'s node document.
6. If *contextDocument* is in quirks mode, then set *document*'s mode to "`quirks`".
7. Otherwise, if *contextDocument* is in limited-quirks mode, then set *document*'s mode to "`limited-quirks`".
8. Create a new HTML parser[p1368] whose allow declarative shadow roots[p1387] is *allowDeclarativeShadowRoots*, and associate it with *document*.
9. Set the parser's fragment context element[p1386] to *context*[p1473].
10. If *contextDocument*'s scripting is disabled[p1152], then set *scriptingMode* to Disabled[p1387].
11. Set the parser's scripting mode[p1387] to *scriptingMode*.
12. Set the state of the HTML parser[p1368]'s tokenization[p1387] stage as follows, switching on *context*[p1473]:

    ↪ **`title`**[p181]
    ↪ **`textarea`**[p606]
    Switch the tokenizer to the RCDATA state[p1388].

    ↪ **`style`**[p207]
    ↪ **`xmp`**[p1531]
    ↪ **`iframe`**[p404]
    ↪ **`noembed`**[p1530]
    ↪ **`noframes`**[p1530]
    Switch the tokenizer to the RAWTEXT state[p1389].

    ↪ **`script`**[p685]
    Switch the tokenizer to the script data state[p1389].

    ↪ **`noscript`**[p703]
    If scripting mode[p1387] is not Disabled[p1387], switch the tokenizer to the RAWTEXT state[p1389]. Otherwise, leave the tokenizer in the data state[p1388].

↪ **plaintext**[p1530]

Switch the tokenizer to the PLAINTEXT state[p1389].

↪ **Any other element**

Leave the tokenizer in the data state[p1388].

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[p1388] in the fragment case, yet they involve far fewer state transitions.

13. Let *root* be the result of creating an element given *document*, "`html`", the HTML namespace, null, null, false, and the result of looking up a custom element registry[p799] given *target*.

14. Append *root* to *document*.

15. Set up the HTML parser[p1368]'s stack of open elements[p1383] so that it contains just the single element *root*.

16. Let *fragment* be the result of creating a document fragment given *target*'s node document.

17. Set the parser's root insertion target[p1386] to *fragment*.

18. If *context*[p1473] is a `template`[p705] element, then push "in template[p1452]" onto the stack of template insertion modes[p1383] so that it is the new current template insertion mode[p1383].

19. Create a start tag token whose name is the local name of *context*[p1473] and whose attributes are the attributes of *context*[p1473].

    Let this start tag token be the start tag token of *context*[p1473]; e.g. for the purposes of determining if it is an HTML integration point[p1417].

20. Reset the parser's insertion mode appropriately[p1383].

    Note

    The parser will reference its fragment context element[p1386] as part of that algorithm.

21. Set the HTML parser[p1368]'s `form` element pointer[p1386] to the nearest node to *context*[p1473] that is a `form`[p532] element (going straight up the ancestor chain, and including the element itself, if it is a `form`[p532] element), if any. (If there is no such `form`[p532] element, the `form` element pointer[p1386] keeps its initial value, null.)

22. Place the *input* into the input stream[p1382] for the HTML parser[p1368] just created. The encoding confidence[p1375] is *irrelevant*.

23. Start the HTML parser[p1368] and let it run until it has consumed all the characters just inserted into the input stream.

24. Return *fragment*.

13.5 Named character references §[p1474]

This table lists the character reference[p1366] 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.

Name	Character(s)	Glyph
Aacute;	U+000C1	Á
Aacute	U+000C1	Á
aacute;	U+000E1	á
aacute	U+000E1	á
Abreve;	U+00102	Ă
abreve;	U+00103	ă
ac;	U+0223E	∾
acd;	U+0223F	∿
acE;	U+0223E U+00333	∾̳
Acirc;	U+000C2	Â

Name	Character(s)	Glyph
Acirc	U+000C2	Â
acirc;	U+000E2	â
acirc	U+000E2	â
acute;	U+000B4	´
acute	U+000B4	´
Acy;	U+00410	А
acy;	U+00430	а
AElig;	U+000C6	Æ
AElig	U+000C6	Æ
aelig;	U+000E6	æ

Name	Character(s)	Glyph
aelig	U+000E6	æ
af;	U+02061	⁡
Afr;	U+1D504	𝔄
afr;	U+1D51E	𝔞
Agrave;	U+000C0	À
Agrave	U+000C0	À
agrave;	U+000E0	à
agrave	U+000E0	à
alefsym;	U+02135	ℵ
aleph;	U+02135	ℵ

Name	Character(s)	Glyph
Alpha;	U+00391	Α
alpha;	U+003B1	α
Amacr;	U+00100	Ā
amacr;	U+00101	ā
amalg;	U+02A3F	⨿
AMP;	U+00026	&
AMP	U+00026	&
amp;	U+00026	&
amp	U+00026	&
And;	U+02A53	⩓
and;	U+02227	∧
andand;	U+02A55	⩕
andd;	U+02A5C	⩜
andslope;	U+02A58	⩘
andv;	U+02A5A	⩚
ang;	U+02220	∠
ange;	U+029A4	⦤
angle;	U+02220	∠
angmsd;	U+02221	∡
angmsdaa;	U+029A8	⦨
angmsdab;	U+029A9	⦩
angmsdac;	U+029AA	⦪
angmsdad;	U+029AB	⦫
angmsdae;	U+029AC	⦬
angmsdaf;	U+029AD	⦭
angmsdag;	U+029AE	⦮
angmsdah;	U+029AF	⦯
angrt;	U+0221F	∟
angrtvb;	U+022BE	⊾
angrtvbd;	U+0299D	⦝
angsph;	U+02222	∢
angst;	U+000C5	Å
angzarr;	U+0237C	⍼
Aogon;	U+00104	Ą
aogon;	U+00105	ą
Aopf;	U+1D538	𝔸
aopf;	U+1D552	𝕒
ap;	U+02248	≈
apacir;	U+02A6F	⩯
apE;	U+02A70	⩰
ape;	U+0224A	≊
apid;	U+0224B	≋
apos;	U+00027	'
ApplyFunction;	U+02061	⁡
approx;	U+02248	≈
approxeq;	U+0224A	≊
Aring;	U+000C5	Å
Aring	U+000C5	Å
aring;	U+000E5	å
aring	U+000E5	å
Ascr;	U+1D49C	𝒜
ascr;	U+1D4B6	𝒶
Assign;	U+02254	≔
ast;	U+0002A	*
asymp;	U+02248	≈
asympeq;	U+0224D	≍
Atilde;	U+000C3	Ã
Atilde	U+000C3	Ã
atilde;	U+000E3	ã
atilde	U+000E3	ã
Auml;	U+000C4	Ä
Auml	U+000C4	Ä
auml;	U+000E4	ä
auml	U+000E4	ä
awconint;	U+02233	∳
awint;	U+02A11	⨑
backcong;	U+0224C	≌
backepsilon;	U+003F6	϶
backprime;	U+02035	‵
backsim;	U+0223D	∽
backsimeq;	U+022CD	⋍
Backslash;	U+02216	∖
Barv;	U+02AE7	⫧
barvee;	U+022BD	⊽
Barwed;	U+02306	⌆
barwed;	U+02305	⌅
barwedge;	U+02305	⌅
bbrk;	U+023B5	⎵
bbrktbrk;	U+023B6	⎶
bcong;	U+0224C	≌
Bcy;	U+00411	Б
bcy;	U+00431	б
bdquo;	U+0201E	„
becaus;	U+02235	∵
Because;	U+02235	∵
because;	U+02235	∵
bemptyv;	U+029B0	⦰
bepsi;	U+003F6	϶
bernou;	U+0212C	ℬ
Bernoullis;	U+0212C	ℬ
Beta;	U+00392	Β

Name	Character(s)	Glyph
beta;	U+003B2	β
beth;	U+02136	ℶ
between;	U+0226C	≬
Bfr;	U+1D505	𝔅
bfr;	U+1D51F	𝔟
bigcap;	U+022C2	⋂
bigcirc;	U+025EF	◯
bigcup;	U+022C3	⋃
bigodot;	U+02A00	⨀
bigoplus;	U+02A01	⨁
bigotimes;	U+02A02	⨂
bigsqcup;	U+02A06	⨆
bigstar;	U+02605	★
bigtriangledown;	U+025BD	▽
bigtriangleup;	U+025B3	△
biguplus;	U+02A04	⨄
bigvee;	U+022C1	⋁
bigwedge;	U+022C0	⋀
bkarow;	U+0290D	⤍
blacklozenge;	U+029EB	⧫
blacksquare;	U+025AA	▪
blacktriangle;	U+025B4	▴
blacktriangledown;	U+025BE	▾
blacktriangleleft;	U+025C2	◂
blacktriangleright;	U+025B8	▸
blank;	U+02423	␣
blk12;	U+02592	▒
blk14;	U+02591	░
blk34;	U+02593	▓
block;	U+02588	█
bne;	U+0003D U+020E5	=⃥
bnequiv;	U+02261 U+020E5	≡⃥
bNot;	U+02AED	⫭
bnot;	U+02310	⌐
Bopf;	U+1D539	𝔹
bopf;	U+1D553	𝕓
bot;	U+022A5	⊥
bottom;	U+022A5	⊥
bowtie;	U+022C8	⋈
boxbox;	U+029C9	⧉
boxDL;	U+02557	╗
boxDl;	U+02556	╖
boxdL;	U+02555	╕
boxdl;	U+02510	┐
boxDR;	U+02554	╔
boxDr;	U+02553	╓
boxdR;	U+02552	╒
boxdr;	U+0250C	┌
boxH;	U+02550	═
boxh;	U+02500	─
boxHD;	U+02566	╦
boxHd;	U+02564	╤
boxhD;	U+02565	╥
boxhd;	U+0252C	┬
boxHU;	U+02569	╩
boxHu;	U+02567	╧
boxhU;	U+02568	╨
boxhu;	U+02534	┴
boxminus;	U+0229F	⊟
boxplus;	U+0229E	⊞
boxtimes;	U+022A0	⊠
boxUL;	U+0255D	╝
boxUl;	U+0255C	╜
boxuL;	U+0255B	╛
boxul;	U+02518	┘
boxUR;	U+0255A	╚
boxUr;	U+02559	╙
boxuR;	U+02558	╘
boxur;	U+02514	└
boxV;	U+02551	║
boxv;	U+02502	│
boxVH;	U+0256C	╬
boxVh;	U+0256B	╫
boxvH;	U+0256A	╪
boxvh;	U+0253C	┼
boxVL;	U+02563	╣
boxVl;	U+02562	╢
boxvL;	U+02561	╡
boxvl;	U+02524	┤
boxVR;	U+02560	╠
boxVr;	U+0255F	╟
boxvR;	U+0255E	╞
boxvr;	U+0251C	├
bprime;	U+02035	‵
Breve;	U+002D8	˘
breve;	U+002D8	˘
brvbar;	U+000A6	¦
brvbar	U+000A6	¦
Bscr;	U+0212C	ℬ
bscr;	U+1D4B7	𝒷
bsemi;	U+0204F	⁏

Name	Character(s)	Glyph
bsim;	U+0223D	∽
bsime;	U+022CD	⋍
bsol;	U+0005C	\
bsolb;	U+029C5	⧅
bsolhsub;	U+027C8	⟈
bull;	U+02022	•
bullet;	U+02022	•
bump;	U+0224E	≎
bumpE;	U+02AAE	⪮
bumpe;	U+0224F	≏
Bumpeq;	U+0224E	≎
bumpeq;	U+0224F	≏
Cacute;	U+00106	Ć
cacute;	U+00107	ć
Cap;	U+022D2	⋒
cap;	U+02229	∩
capand;	U+02A44	⩄
capbrcup;	U+02A49	⩉
capcap;	U+02A4B	⩋
capcup;	U+02A47	⩇
capdot;	U+02A40	⩀
CapitalDifferentialD;	U+02145	ⅅ
caps;	U+02229 U+0FE00	∩
caret;	U+02041	⁁
caron;	U+002C7	ˇ
Cayleys;	U+0212D	ℭ
ccaps;	U+02A4D	⩍
Ccaron;	U+0010C	Č
ccaron;	U+0010D	č
Ccedil;	U+000C7	Ç
Ccedil	U+000C7	Ç
ccedil;	U+000E7	ç
ccedil	U+000E7	ç
Ccirc;	U+00108	Ĉ
ccirc;	U+00109	ĉ
Cconint;	U+02230	∰
ccups;	U+02A4C	⩌
ccupssm;	U+02A50	⩐
Cdot;	U+0010A	Ċ
cdot;	U+0010B	ċ
cedil;	U+000B8	¸
cedil	U+000B8	¸
Cedilla;	U+000B8	¸
cemptyv;	U+029B2	⦲
cent;	U+000A2	¢
cent	U+000A2	¢
CenterDot;	U+000B7	·
centerdot;	U+000B7	·
Cfr;	U+0212D	ℭ
cfr;	U+1D520	𝔠
CHcy;	U+00427	Ч
chcy;	U+00447	ч
check;	U+02713	✓
checkmark;	U+02713	✓
Chi;	U+003A7	Χ
chi;	U+003C7	χ
cir;	U+025CB	○
circ;	U+002C6	ˆ
circeq;	U+02257	≗
circlearrowleft;	U+021BA	↺
circlearrowright;	U+021BB	↻
circledast;	U+0229B	⊛
circledcirc;	U+0229A	⊚
circleddash;	U+0229D	⊝
CircleDot;	U+02299	⊙
circledR;	U+000AE	®
circledS;	U+024C8	Ⓢ
CircleMinus;	U+02296	⊖
CirclePlus;	U+02295	⊕
CircleTimes;	U+02297	⊗
cirE;	U+029C3	⧃
cire;	U+02257	≗
cirfnint;	U+02A10	⨐
cirmid;	U+02AEF	⫯
cirscir;	U+029C2	⧂
ClockwiseContourIntegral;	U+02232	∲
CloseCurlyDoubleQuote;	U+0201D	”
CloseCurlyQuote;	U+02019	’
clubs;	U+02663	♣
clubsuit;	U+02663	♣
Colon;	U+02237	∷
colon;	U+0003A	:
Colone;	U+02A74	⩴
colone;	U+02254	≔
coloneq;	U+02254	≔
comma;	U+0002C	,
commat;	U+00040	@
comp;	U+02201	∁
compfn;	U+02218	∘
complement;	U+02201	∁
complexes;	U+02102	ℂ

Name	Character(s)	Glyph
cong;	U+02245	≅
congdot;	U+02A6D	⩭
Congruent;	U+02261	≡
Conint;	U+0222F	∯
conint;	U+0222E	∮
ContourIntegral;	U+0222E	∮
Copf;	U+02102	ℂ
copf;	U+1D554	𝕔
coprod;	U+02210	∐
Coproduct;	U+02210	∐
COPY;	U+000A9	©
COPY	U+000A9	©
copy;	U+000A9	©
copy	U+000A9	©
copysr;	U+02117	℗
CounterClockwiseContourIntegral;	U+02233	∳
crarr;	U+021B5	↵
Cross;	U+02A2F	×
cross;	U+02717	✗
Cscr;	U+1D49E	𝒞
cscr;	U+1D4B8	𝒸
csub;	U+02ACF	⫏
csube;	U+02AD1	⫑
csup;	U+02AD0	⫐
csupe;	U+02AD2	⫒
ctdot;	U+022EF	⋯
cudarrl;	U+02938	⤸
cudarrr;	U+02935	⤵
cuepr;	U+022DE	⋞
cuesc;	U+022DF	⋟
cularr;	U+021B6	↶
cularrp;	U+0293D	⤽
Cup;	U+022D3	⋓
cup;	U+0222A	∪
cupbrcap;	U+02A48	⩈
CupCap;	U+0224D	≍
cupcap;	U+02A46	⩆
cupcup;	U+02A4A	⩊
cupdot;	U+0228D	⊍
cupor;	U+02A45	⩅
cups;	U+0222A U+0FE00	∪
curarr;	U+021B7	↷
curarrm;	U+0293C	⤼
curlyeqprec;	U+022DE	⋞
curlyeqsucc;	U+022DF	⋟
curlyvee;	U+022CE	⋎
curlywedge;	U+022CF	⋏
curren;	U+000A4	¤
curren	U+000A4	¤
curvearrowleft;	U+021B6	↶
curvearrowright;	U+021B7	↷
cuvee;	U+022CE	⋎
cuwed;	U+022CF	⋏
cwconint;	U+02232	∲
cwint;	U+02231	∱
cylcty;	U+0232D	⌭
Dagger;	U+02021	‡
dagger;	U+02020	†
daleth;	U+02138	ℸ
Darr;	U+021A1	↡
dArr;	U+021D3	⇓
darr;	U+02193	↓
dash;	U+02010	‐
Dashv;	U+02AE4	⫤
dashv;	U+022A3	⊣
dbkarow;	U+0290F	⤏
dblac;	U+002DD	˝
Dcaron;	U+0010E	Ď
dcaron;	U+0010F	ď
Dcy;	U+00414	Д
dcy;	U+00434	д
DD;	U+02145	ⅅ
dd;	U+02146	ⅆ
ddagger;	U+02021	‡
ddarr;	U+021CA	⇊
DDotrahd;	U+02911	⤑
ddotseq;	U+02A77	⩷
deg;	U+000B0	°
deg	U+000B0	°
Del;	U+02207	∇
Delta;	U+00394	Δ
delta;	U+003B4	δ
demptyv;	U+029B1	⦱
dfisht;	U+0297F	⥿
Dfr;	U+1D507	𝔇
dfr;	U+1D521	𝔡
dHar;	U+02965	⥥
dharl;	U+021C3	⇃
dharr;	U+021C2	⇂
DiacriticalAcute;	U+000B4	´
DiacriticalDot;	U+002D9	˙

Name	Character(s)	Glyph
DiacriticalDoubleAcute;	U+002DD	˝
DiacriticalGrave;	U+00060	`
DiacriticalTilde;	U+002DC	˜
diam;	U+022C4	⋄
Diamond;	U+022C4	⋄
diamond;	U+022C4	⋄
diamondsuit;	U+02666	♦
diams;	U+02666	♦
die;	U+000A8	¨
DifferentialD;	U+02146	ⅆ
digamma;	U+003DD	ϝ
disin;	U+022F2	⋲
div;	U+000F7	÷
divide;	U+000F7	÷
divide	U+000F7	÷
divideontimes;	U+022C7	⋇
divonx;	U+022C7	⋇
DJcy;	U+00402	Ђ
djcy;	U+00452	ђ
dlcorn;	U+0231E	⌞
dlcrop;	U+0230D	⌍
dollar;	U+00024	$
Dopf;	U+1D53B	𝔻
dopf;	U+1D555	𝕕
Dot;	U+000A8	¨
dot;	U+002D9	˙
DotDot;	U+020DC	⃜
doteq;	U+02250	≐
doteqdot;	U+02251	≑
DotEqual;	U+02250	≐
dotminus;	U+02238	∸
dotplus;	U+02214	∔
dotsquare;	U+022A1	⊡
doublebarwedge;	U+02306	⌆
DoubleContourIntegral;	U+0222F	∯
DoubleDot;	U+000A8	¨
DoubleDownArrow;	U+021D3	⇓
DoubleLeftArrow;	U+021D0	⇐
DoubleLeftRightArrow;	U+021D4	⇔
DoubleLeftTee;	U+02AE4	⫤
DoubleLongLeftArrow;	U+027F8	⟸
DoubleLongLeftRightArrow;	U+027FA	⟺
DoubleLongRightArrow;	U+027F9	⟹
DoubleRightArrow;	U+021D2	⇒
DoubleRightTee;	U+022A8	⊨
DoubleUpArrow;	U+021D1	⇑
DoubleUpDownArrow;	U+021D5	⇕
DoubleVerticalBar;	U+02225	∥
DownArrow;	U+02193	↓
Downarrow;	U+021D3	⇓
downarrow;	U+02193	↓
DownArrowBar;	U+02913	⤓
DownArrowUpArrow;	U+021F5	⇵
DownBreve;	U+00311	◌̑
downdownarrows;	U+021CA	⇊
downharpoonleft;	U+021C3	⇃
downharpoonright;	U+021C2	⇂
DownLeftRightVector;	U+02950	⥐
DownLeftTeeVector;	U+0295E	⥞
DownLeftVector;	U+021BD	↽
DownLeftVectorBar;	U+02956	⥖
DownRightTeeVector;	U+0295F	⥟
DownRightVector;	U+021C1	⇁
DownRightVectorBar;	U+02957	⥗
DownTee;	U+022A4	⊤
DownTeeArrow;	U+021A7	↧
drbkarow;	U+02910	⤐
drcorn;	U+0231F	⌟
drcrop;	U+0230C	⌌
Dscr;	U+1D49F	𝒟
dscr;	U+1D4B9	𝒹
DScy;	U+00405	Ѕ
dscy;	U+00455	ѕ
dsol;	U+029F6	⧶
Dstrok;	U+00110	Đ
dstrok;	U+00111	đ
dtdot;	U+022F1	⋱
dtri;	U+025BF	▿
dtrif;	U+025BE	▾
duarr;	U+021F5	⇵
duhar;	U+0296F	⥯
dwangle;	U+029A6	⦦
DZcy;	U+0040F	Џ
dzcy;	U+0045F	џ
dzigrarr;	U+027FF	⟿
Eacute;	U+000C9	É
Eacute	U+000C9	É
eacute;	U+000E9	é
eacute	U+000E9	é
easter;	U+02A6E	⩮
Ecaron;	U+0011A	Ě

Name	Character(s)	Glyph
ecaron;	U+0011B	ě
ecir;	U+02256	≖
Ecirc;	U+000CA	Ê
Ecirc	U+000CA	Ê
ecirc;	U+000EA	ê
ecirc	U+000EA	ê
ecolon;	U+02255	≕
Ecy;	U+0042D	Э
ecy;	U+0044D	э
eDDot;	U+02A77	⩷
Edot;	U+00116	Ė
eDot;	U+02251	≑
edot;	U+00117	ė
ee;	U+02147	ⅇ
efDot;	U+02252	≒
Efr;	U+1D508	𝔈
efr;	U+1D522	𝔢
eg;	U+02A9A	⪚
Egrave;	U+000C8	È
Egrave	U+000C8	È
egrave;	U+000E8	è
egrave	U+000E8	è
egs;	U+02A96	⪖
egsdot;	U+02A98	⪘
el;	U+02A99	⪙
Element;	U+02208	∈
elinters;	U+023E7	⏧
ell;	U+02113	ℓ
els;	U+02A95	⪕
elsdot;	U+02A97	⪗
Emacr;	U+00112	Ē
emacr;	U+00113	ē
empty;	U+02205	∅
emptyset;	U+02205	∅
EmptySmallSquare;	U+025FB	□
emptyv;	U+02205	∅
EmptyVerySmallSquare;	U+025AB	▫
emsp;	U+02003	
emsp13;	U+02004	
emsp14;	U+02005	
ENG;	U+0014A	Ŋ
eng;	U+0014B	ŋ
ensp;	U+02002	
Eogon;	U+00118	Ę
eogon;	U+00119	ę
Eopf;	U+1D53C	𝔼
eopf;	U+1D556	𝕖
epar;	U+022D5	⋕
eparsl;	U+029E3	⧣
eplus;	U+02A71	⩱
epsi;	U+003B5	ε
Epsilon;	U+00395	Ε
epsilon;	U+003B5	ε
epsiv;	U+003F5	ϵ
eqcirc;	U+02256	≖
eqcolon;	U+02255	≕
eqsim;	U+02242	≂
eqslantgtr;	U+02A96	⪖
eqslantless;	U+02A95	⪕
Equal;	U+02A75	⩵
equals;	U+0003D	=
EqualTilde;	U+02242	≂
equest;	U+0225F	≟
Equilibrium;	U+021CC	⇌
equiv;	U+02261	≡
equivDD;	U+02A78	⩸
eqvparsl;	U+029E5	⧥
erarr;	U+02971	⥱
erDot;	U+02253	≓
Escr;	U+02130	ℰ
escr;	U+0212F	ℯ
esdot;	U+02250	≐
Esim;	U+02A73	⩳
esim;	U+02242	≂
Eta;	U+00397	Η
eta;	U+003B7	η
ETH;	U+000D0	Ð
ETH	U+000D0	Ð
eth;	U+000F0	ð
eth	U+000F0	ð
Euml;	U+000CB	Ë
Euml	U+000CB	Ë
euml;	U+000EB	ë
euml	U+000EB	ë
euro;	U+020AC	€
excl;	U+00021	!
exist;	U+02203	∃
Exists;	U+02203	∃
expectation;	U+02130	ℰ
ExponentialE;	U+02147	ⅇ
exponentiale;	U+02147	ⅇ

Name	Character(s)	Glyph
fallingdotseq;	U+02252	≒
Fcy;	U+00424	Ф
fcy;	U+00444	ф
female;	U+02640	♀
ffilig;	U+0FB03	ffi
fflig;	U+0FB00	ff
ffllig;	U+0FB04	ffl
Ffr;	U+1D509	𝔉
ffr;	U+1D523	𝔣
filig;	U+0FB01	fi
FilledSmallSquare;	U+025FC	◼
FilledVerySmallSquare;	U+025AA	▪
fjlig;	U+00066 U+0006A	fj
flat;	U+0266D	♭
fllig;	U+0FB02	fl
fltns;	U+025B1	▱
fnof;	U+00192	ƒ
Fopf;	U+1D53D	𝔽
fopf;	U+1D557	𝕗
ForAll;	U+02200	∀
forall;	U+02200	∀
fork;	U+022D4	⋔
forkv;	U+02AD9	⫙
Fouriertrf;	U+02131	ℱ
fpartint;	U+02A0D	⨍
frac12;	U+000BD	½
frac12	U+000BD	½
frac13;	U+02153	⅓
frac14;	U+000BC	¼
frac14	U+000BC	¼
frac15;	U+02155	⅕
frac16;	U+02159	⅙
frac18;	U+0215B	⅛
frac23;	U+02154	⅔
frac25;	U+02156	⅖
frac34;	U+000BE	¾
frac34	U+000BE	¾
frac35;	U+02157	⅗
frac38;	U+0215C	⅜
frac45;	U+02158	⅘
frac56;	U+0215A	⅚
frac58;	U+0215D	⅝
frac78;	U+0215E	⅞
frasl;	U+02044	⁄
frown;	U+02322	⌢
Fscr;	U+02131	ℱ
fscr;	U+1D4BB	𝒻
gacute;	U+001F5	ǵ
Gamma;	U+00393	Γ
gamma;	U+003B3	γ
Gammad;	U+003DC	Ϝ
gammad;	U+003DD	ϝ
gap;	U+02A86	⪆
Gbreve;	U+0011E	Ğ
gbreve;	U+0011F	ğ
Gcedil;	U+00122	Ģ
Gcirc;	U+0011C	Ĝ
gcirc;	U+0011D	ĝ
Gcy;	U+00413	Г
gcy;	U+00433	г
Gdot;	U+00120	Ġ
gdot;	U+00121	ġ
gE;	U+02267	≧
ge;	U+02265	≥
gEl;	U+02A8C	⪌
gel;	U+022DB	⋛
geq;	U+02265	≥
geqq;	U+02267	≧
geqslant;	U+02A7E	⩾
ges;	U+02A7E	⩾
gescc;	U+02AA9	⪩
gesdot;	U+02A80	⪀
gesdoto;	U+02A82	⪂
gesdotol;	U+02A84	⪄
gesl;	U+022DB U+0FE00	⋛
gesles;	U+02A94	⪔
Gfr;	U+1D50A	𝔊
gfr;	U+1D524	𝔤
Gg;	U+022D9	⋙
gg;	U+0226B	≫
ggg;	U+022D9	⋙
gimel;	U+02137	ℷ
GJcy;	U+00403	Ѓ
gjcy;	U+00453	ѓ
gl;	U+02277	≷
gla;	U+02AA5	⪥
glE;	U+02A92	⪒
glj;	U+02AA4	⪤
gnap;	U+02A8A	⪊
gnapprox;	U+02A8A	⪊
gnE;	U+02269	≩

Name	Character(s)	Glyph
gne;	U+02A88	⪈
gneq;	U+02A88	⪈
gneqq;	U+02269	≩
gnsim;	U+022E7	⋧
Gopf;	U+1D53E	𝔾
gopf;	U+1D558	𝕘
grave;	U+00060	`
GreaterEqual;	U+02265	≥
GreaterEqualLess;	U+022DB	⋛
GreaterFullEqual;	U+02267	≧
GreaterGreater;	U+02AA2	⪢
GreaterLess;	U+02277	≷
GreaterSlantEqual;	U+02A7E	⩾
GreaterTilde;	U+02273	≳
Gscr;	U+1D4A2	𝒢
gscr;	U+0210A	ℊ
gsim;	U+02273	≳
gsime;	U+02A8E	⪎
gsiml;	U+02A90	⪐
GT;	U+0003E	>
GT	U+0003E	>
Gt;	U+0226B	≫
gt;	U+0003E	>
gt	U+0003E	>
gtcc;	U+02AA7	⪧
gtcir;	U+02A7A	⩺
gtdot;	U+022D7	⋗
gtlPar;	U+02995	⦕
gtquest;	U+02A7C	⩼
gtrapprox;	U+02A86	⪆
gtrarr;	U+02978	⥸
gtrdot;	U+022D7	⋗
gtreqless;	U+022DB	⋛
gtreqqless;	U+02A8C	⪌
gtrless;	U+02277	≷
gtrsim;	U+02273	≳
gvertneqq;	U+02269 U+0FE00	≩
gvnE;	U+02269 U+0FE00	≩
Hacek;	U+002C7	ˇ
hairsp;	U+0200A	
half;	U+000BD	½
hamilt;	U+0210B	ℋ
HARDcy;	U+0042A	Ъ
hardcy;	U+0044A	ъ
hArr;	U+021D4	⇔
harr;	U+02194	↔
harrcir;	U+02948	⥈
harrw;	U+021AD	↭
Hat;	U+0005E	^
hbar;	U+0210F	ℏ
Hcirc;	U+00124	Ĥ
hcirc;	U+00125	ĥ
hearts;	U+02665	♥
heartsuit;	U+02665	♥
hellip;	U+02026	…
hercon;	U+022B9	⊹
Hfr;	U+0210C	ℌ
hfr;	U+1D525	𝔥
HilbertSpace;	U+0210B	ℋ
hksearow;	U+02925	⤥
hkswarow;	U+02926	⤦
hoarr;	U+021FF	⇿
homtht;	U+0223B	∻
hookleftarrow;	U+021A9	↩
hookrightarrow;	U+021AA	↪
Hopf;	U+0210D	ℍ
hopf;	U+1D559	𝕙
horbar;	U+02015	―
HorizontalLine;	U+02500	─
Hscr;	U+0210B	ℋ
hscr;	U+1D4BD	𝒽
hslash;	U+0210F	ℏ
Hstrok;	U+00126	Ħ
hstrok;	U+00127	ħ
HumpDownHump;	U+0224E	≎
HumpEqual;	U+0224F	≏
hybull;	U+02043	⁃
hyphen;	U+02010	‐
Iacute;	U+000CD	Í
Iacute	U+000CD	Í
iacute;	U+000ED	í
iacute	U+000ED	í
ic;	U+02063	
Icirc;	U+000CE	Î
Icirc	U+000CE	Î
icirc;	U+000EE	î
icirc	U+000EE	î
Icy;	U+00418	И
icy;	U+00438	и
Idot;	U+00130	İ
IEcy;	U+00415	Е

Name	Character(s)	Glyph
iecy;	U+00435	е
iexcl;	U+000A1	¡
iexcl	U+000A1	¡
iff;	U+021D4	⇔
Ifr;	U+02111	ℑ
ifr;	U+1D526	𝔦
Igrave;	U+000CC	Ì
Igrave	U+000CC	Ì
igrave;	U+000EC	ì
igrave	U+000EC	ì
ii;	U+02148	ⅈ
iiiint;	U+02A0C	⨌
iiint;	U+0222D	∭
iinfin;	U+029DC	⧜
iiota;	U+02129	℩
IJlig;	U+00132	Ĳ
ijlig;	U+00133	ĳ
Im;	U+02111	ℑ
Imacr;	U+0012A	Ī
imacr;	U+0012B	ī
image;	U+02111	ℑ
ImaginaryI;	U+02148	ⅈ
imagline;	U+02110	ℐ
imagpart;	U+02111	ℑ
imath;	U+00131	ı
imof;	U+022B7	⊷
imped;	U+001B5	Ƶ
Implies;	U+021D2	⇒
in;	U+02208	∈
incare;	U+02105	℅
infin;	U+0221E	∞
infintie;	U+029DD	⧝
inodot;	U+00131	ı
Int;	U+0222C	∬
int;	U+0222B	∫
intcal;	U+022BA	⊺
integers;	U+02124	ℤ
Integral;	U+0222B	∫
intercal;	U+022BA	⊺
Intersection;	U+022C2	⋂
intlarhk;	U+02A17	⨗
intprod;	U+02A3C	⨼
InvisibleComma;	U+02063	
InvisibleTimes;	U+02062	
IOcy;	U+00401	Ё
iocy;	U+00451	ё
Iogon;	U+0012E	Į
iogon;	U+0012F	į
Iopf;	U+1D540	𝕀
iopf;	U+1D55A	𝕚
Iota;	U+00399	Ι
iota;	U+003B9	ι
iprod;	U+02A3C	⨼
iquest;	U+000BF	¿
iquest	U+000BF	¿
Iscr;	U+02110	ℐ
iscr;	U+1D4BE	𝒾
isin;	U+02208	∈
isindot;	U+022F5	⋵
isinE;	U+022F9	⋹
isins;	U+022F4	⋴
isinsv;	U+022F3	⋳
isinv;	U+02208	∈
it;	U+02062	
Itilde;	U+00128	Ĩ
itilde;	U+00129	ĩ
Iukcy;	U+00406	І
iukcy;	U+00456	і
Iuml;	U+000CF	Ï
Iuml	U+000CF	Ï
iuml;	U+000EF	ï
iuml	U+000EF	ï
Jcirc;	U+00134	Ĵ
jcirc;	U+00135	ĵ
Jcy;	U+00419	Й
jcy;	U+00439	й
Jfr;	U+1D50D	𝔍
jfr;	U+1D527	𝔧
jmath;	U+00237	ȷ
Jopf;	U+1D541	𝕁
jopf;	U+1D55B	𝕛
Jscr;	U+1D4A5	𝒥
jscr;	U+1D4BF	𝒿
Jsercy;	U+00408	Ј
jsercy;	U+00458	ј
Jukcy;	U+00404	Є
jukcy;	U+00454	є
Kappa;	U+0039A	Κ
kappa;	U+003BA	κ
kappav;	U+003F0	ϰ
Kcedil;	U+00136	Ķ

Name	Character(s)	Glyph
kcedil;	U+00137	ķ
Kcy;	U+0041A	К
kcy;	U+0043A	к
Kfr;	U+1D50E	𝔎
kfr;	U+1D528	𝔨
kgreen;	U+00138	ĸ
KHcy;	U+00425	Х
khcy;	U+00445	х
KJcy;	U+0040C	Ќ
kjcy;	U+0045C	ќ
Kopf;	U+1D542	𝕂
kopf;	U+1D55C	𝕜
Kscr;	U+1D4A6	𝒦
kscr;	U+1D4C0	𝓀
lAarr;	U+021DA	⇚
Lacute;	U+00139	Ĺ
lacute;	U+0013A	ĺ
laemptyv;	U+029B4	⦴
lagran;	U+02112	ℒ
Lambda;	U+0039B	Λ
lambda;	U+003BB	λ
Lang;	U+027EA	⟪
lang;	U+027E8	⟨
langd;	U+02991	⦑
langle;	U+027E8	⟨
lap;	U+02A85	⪅
Laplacetrf;	U+02112	ℒ
laquo;	U+000AB	«
laquo	U+000AB	«
Larr;	U+0219E	↞
lArr;	U+021D0	⇐
larr;	U+02190	←
larrb;	U+021E4	⇤
larrbfs;	U+0291F	⤟
larrfs;	U+0291D	⤝
larrhk;	U+021A9	↩
larrlp;	U+021AB	↫
larrpl;	U+02939	⤹
larrsim;	U+02973	⥳
larrtl;	U+021A2	↢
lat;	U+02AAB	⪫
lAtail;	U+0291B	⤛
latail;	U+02919	⤙
late;	U+02AAD	⪭
lates;	U+02AAD U+0FE00	⪭
lBarr;	U+0290E	⤎
lbarr;	U+0290C	⤌
lbbrk;	U+02772	❲
lbrace;	U+0007B	{
lbrack;	U+0005B	[
lbrke;	U+0298B	⦋
lbrksld;	U+0298F	⦏
lbrkslu;	U+0298D	⦍
Lcaron;	U+0013D	Ľ
lcaron;	U+0013E	ľ
Lcedil;	U+0013B	Ļ
lcedil;	U+0013C	ļ
lceil;	U+02308	⌈
lcub;	U+0007B	{
Lcy;	U+0041B	Л
lcy;	U+0043B	л
ldca;	U+02936	⤶
ldquo;	U+0201C	“
ldquor;	U+0201E	„
ldrdhar;	U+02967	⥧
ldrushar;	U+0294B	⥋
ldsh;	U+021B2	↲
lE;	U+02266	≦
le;	U+02264	≤
LeftAngleBracket;	U+027E8	⟨
LeftArrow;	U+02190	←
Leftarrow;	U+021D0	⇐
leftarrow;	U+02190	←
LeftArrowBar;	U+021E4	⇤
LeftArrowRightArrow;	U+021C6	⇆
leftarrowtail;	U+021A2	↢
LeftCeiling;	U+02308	⌈
LeftDoubleBracket;	U+027E6	⟦
LeftDownTeeVector;	U+02961	⥡
LeftDownVector;	U+021C3	⇃
LeftDownVectorBar;	U+02959	⥙
LeftFloor;	U+0230A	⌊
leftharpoondown;	U+021BD	↽
leftharpoonup;	U+021BC	↼
leftleftarrows;	U+021C7	⇇
LeftRightArrow;	U+02194	↔
Leftrightarrow;	U+021D4	⇔
leftrightarrow;	U+02194	↔
leftrightarrows;	U+021C6	⇆
leftrightharpoons;	U+021CB	⇋
leftrightsquigarrow;	U+021AD	↭

Name	Character(s)	Glyph
LeftRightVector;	U+0294E	⥎
LeftTee;	U+022A3	⊣
LeftTeeArrow;	U+021A4	↤
LeftTeeVector;	U+0295A	⥚
leftthreetimes;	U+022CB	⋋
LeftTriangle;	U+022B2	⊲
LeftTriangleBar;	U+029CF	⧏
LeftTriangleEqual;	U+022B4	⊴
LeftUpDownVector;	U+02951	⥑
LeftUpTeeVector;	U+02960	⥠
LeftUpVector;	U+021BF	↿
LeftUpVectorBar;	U+02958	⥘
LeftVector;	U+021BC	↼
LeftVectorBar;	U+02952	⥒
lEg;	U+02A8B	⪋
leg;	U+022DA	⋚
leq;	U+02264	≤
leqq;	U+02266	≦
leqslant;	U+02A7D	⩽
les;	U+02A7D	⩽
lescc;	U+02AA8	⪨
lesdot;	U+02A7F	⩿
lesdoto;	U+02A81	⪁
lesdotor;	U+02A83	⪃
lesg;	U+022DA U+0FE00	⋚
lesges;	U+02A93	⪓
lessapprox;	U+02A85	⪅
lessdot;	U+022D6	⋖
lesseqgtr;	U+022DA	⋚
lesseqqgtr;	U+02A8B	⪋
LessEqualGreater;	U+022DA	⋚
LessFullEqual;	U+02266	≦
LessGreater;	U+02276	≶
lessgtr;	U+02276	≶
LessLess;	U+02AA1	⪡
lesssim;	U+02272	≲
LessSlantEqual;	U+02A7D	⩽
LessTilde;	U+02272	≲
lfisht;	U+0297C	⥼
lfloor;	U+0230A	⌊
Lfr;	U+1D50F	𝔏
lfr;	U+1D529	𝔩
lg;	U+02276	≶
lgE;	U+02A91	⪑
lHar;	U+02962	⥢
lhard;	U+021BD	↽
lharu;	U+021BC	↼
lharul;	U+0296A	⥪
lhblk;	U+02584	▄
LJcy;	U+00409	Љ
ljcy;	U+00459	љ
Ll;	U+022D8	⋘
ll;	U+0226A	≪
llarr;	U+021C7	⇇
llcorner;	U+0231E	⌞
Lleftarrow;	U+021DA	⇚
llhard;	U+0296B	⥫
lltri;	U+025FA	◺
Lmidot;	U+0013F	Ŀ
lmidot;	U+00140	ŀ
lmoust;	U+023B0	⎰
lmoustache;	U+023B0	⎰
lnap;	U+02A89	⪉
lnapprox;	U+02A89	⪉
lnE;	U+02268	≨
lne;	U+02A87	⪇
lneq;	U+02A87	⪇
lneqq;	U+02268	≨
lnsim;	U+022E6	⋦
loang;	U+027EC	⟬
loarr;	U+021FD	⇽
lobrk;	U+027E6	⟦
LongLeftArrow;	U+027F5	⟵
Longleftarrow;	U+027F8	⟸
longleftarrow;	U+027F5	⟵
LongLeftRightArrow;	U+027F7	⟷
Longleftrightarrow;	U+027FA	⟺
longleftrightarrow;	U+027F7	⟷
longmapsto;	U+027FC	⟼
LongRightArrow;	U+027F6	⟶
Longrightarrow;	U+027F9	⟹
longrightarrow;	U+027F6	⟶
looparrowleft;	U+021AB	↫
looparrowright;	U+021AC	↬
lopar;	U+02985	⦅
Lopf;	U+1D543	𝕃
lopf;	U+1D55D	𝕝
loplus;	U+02A2D	⨭
lotimes;	U+02A34	⨴
lowast;	U+02217	∗
lowbar;	U+0005F	_

Name	Character(s)	Glyph
LowerLeftArrow;	U+02199	↙
LowerRightArrow;	U+02198	↘
loz;	U+025CA	◊
lozenge;	U+025CA	◊
lozf;	U+029EB	⧫
lpar;	U+00028	(
lparlt;	U+02993	⦓
lrarr;	U+021C6	⇆
lrcorner;	U+0231F	⌟
lrhar;	U+021CB	⇋
lrhard;	U+0296D	⥭
lrm;	U+0200E	
lrtri;	U+022BF	⊿
lsaquo;	U+02039	‹
Lscr;	U+02112	ℒ
lscr;	U+1D4C1	𝓁
Lsh;	U+021B0	↰
lsh;	U+021B0	↰
lsim;	U+02272	≲
lsime;	U+02A8D	⪍
lsimg;	U+02A8F	⪏
lsqb;	U+0005B	[
lsquo;	U+02018	‘
lsquor;	U+0201A	‚
Lstrok;	U+00141	Ł
lstrok;	U+00142	ł
LT;	U+0003C	<
LT	U+0003C	<
Lt;	U+0226A	≪
lt;	U+0003C	<
lt	U+0003C	<
ltcc;	U+02AA6	⪦
ltcir;	U+02A79	⩹
ltdot;	U+022D6	⋖
lthree;	U+022CB	⋋
ltimes;	U+022C9	⋉
ltlarr;	U+02976	⥶
ltquest;	U+02A7B	⩻
ltri;	U+025C3	◃
ltrie;	U+022B4	⊴
ltrif;	U+025C2	◂
ltrPar;	U+02996	⦖
lurdshar;	U+0294A	⥊
luruhar;	U+02966	⥦
lvertneqq;	U+02268 U+0FE00	≨
lvnE;	U+02268 U+0FE00	≨
macr;	U+000AF	¯
macr	U+000AF	¯
male;	U+02642	♂
malt;	U+02720	✠
maltese;	U+02720	✠
Map;	U+02905	⤅
map;	U+021A6	↦
mapsto;	U+021A6	↦
mapstodown;	U+021A7	↧
mapstoleft;	U+021A4	↤
mapstoup;	U+021A5	↥
marker;	U+025AE	▮
mcomma;	U+02A29	⨩
Mcy;	U+0041C	М
mcy;	U+0043C	м
mdash;	U+02014	—
mDDot;	U+0223A	∺
measuredangle;	U+02221	∡
MediumSpace;	U+0205F	
Mellintrf;	U+02133	ℳ
Mfr;	U+1D510	𝔐
mfr;	U+1D52A	𝔪
mho;	U+02127	℧
micro;	U+000B5	µ
micro	U+000B5	µ
mid;	U+02223	∣
midast;	U+0002A	*
midcir;	U+02AF0	⫰
middot;	U+000B7	·
middot	U+000B7	·
minus;	U+02212	−
minusb;	U+0229F	⊟
minusd;	U+02238	∸
minusdu;	U+02A2A	⨪
MinusPlus;	U+02213	∓
mlcp;	U+02ADB	⫛
mldr;	U+02026	…
mnplus;	U+02213	∓
models;	U+022A7	⊧
Mopf;	U+1D544	𝕄
mopf;	U+1D55E	𝕞
mp;	U+02213	∓
Mscr;	U+02133	ℳ
mscr;	U+1D4C2	𝓂
mstpos;	U+0223E	∾

Name	Character(s)	Glyph
Mu;	U+0039C	Μ
mu;	U+003BC	μ
multimap;	U+022B8	⊸
mumap;	U+022B8	⊸
nabla;	U+02207	∇
Nacute;	U+00143	Ń
nacute;	U+00144	ń
nang;	U+02220 U+020D2	∠⃒
nap;	U+02249	≉
napE;	U+02A70 U+00338	⩰̸
napid;	U+0224B U+00338	≋̸
napos;	U+00149	ŉ
napprox;	U+02249	≉
natur;	U+0266E	♮
natural;	U+0266E	♮
naturals;	U+02115	ℕ
nbsp;	U+000A0	
nbsp	U+000A0	
nbump;	U+0224E U+00338	≎̸
nbumpe;	U+0224F U+00338	≏̸
ncap;	U+02A43	⩃
Ncaron;	U+00147	Ň
ncaron;	U+00148	ň
Ncedil;	U+00145	Ņ
ncedil;	U+00146	ņ
ncong;	U+02247	≇
ncongdot;	U+02A6D U+00338	⩭̸
ncup;	U+02A42	⩂
Ncy;	U+0041D	Н
ncy;	U+0043D	н
ndash;	U+02013	–
ne;	U+02260	≠
nearhk;	U+02924	⤤
neArr;	U+021D7	⇗
nearr;	U+02197	↗
nearrow;	U+02197	↗
nedot;	U+02250 U+00338	≐̸
NegativeMediumSpace;	U+0200B	
NegativeThickSpace;	U+0200B	
NegativeThinSpace;	U+0200B	
NegativeVeryThinSpace;	U+0200B	
nequiv;	U+02262	≢
nesear;	U+02928	⤨
nesim;	U+02242 U+00338	≂̸
NestedGreaterGreater;	U+0226B	≫
NestedLessLess;	U+0226A	≪
NewLine;	U+0000A	
nexist;	U+02204	∄
nexists;	U+02204	∄
Nfr;	U+1D511	𝔑
nfr;	U+1D52B	𝔫
ngE;	U+02267 U+00338	≧̸
nge;	U+02271	≱
ngeq;	U+02271	≱
ngeqq;	U+02267 U+00338	≧̸
ngeqslant;	U+02A7E U+00338	⩾̸
nges;	U+02A7E U+00338	⩾̸
nGg;	U+022D9 U+00338	⋙̸
ngsim;	U+02275	≵
nGt;	U+0226B U+020D2	≫⃒
ngt;	U+0226F	≯
ngtr;	U+0226F	≯
nGtv;	U+0226B U+00338	≫̸
nhArr;	U+021CE	⇎
nharr;	U+021AE	↮
nhpar;	U+02AF2	⫲
ni;	U+0220B	∋
nis;	U+022FC	⋼
nisd;	U+022FA	⋺
niv;	U+0220B	∋
NJcy;	U+0040A	Њ
njcy;	U+0045A	њ
nlArr;	U+021CD	⇍
nlarr;	U+0219A	↚
nldr;	U+02025	‥
nlE;	U+02266 U+00338	≦̸
nle;	U+02270	≰
nLeftarrow;	U+021CD	⇍
nleftarrow;	U+0219A	↚
nLeftrightarrow;	U+021CE	⇎
nleftrightarrow;	U+021AE	↮
nleq;	U+02270	≰
nleqq;	U+02266 U+00338	≦̸
nleqslant;	U+02A7D U+00338	⩽̸
nles;	U+02A7D U+00338	⩽̸
nless;	U+0226E	≮
nLl;	U+022D8 U+00338	⋘̸
nlsim;	U+02274	≴
nLt;	U+0226A U+020D2	≪⃒
nlt;	U+0226E	≮
nltri;	U+022EA	⋪
nltrie;	U+022EC	⋬
nLtv;	U+0226A U+00338	≪̸
nmid;	U+02224	∤
NoBreak;	U+02060	
NonBreakingSpace;	U+000A0	
Nopf;	U+02115	ℕ
nopf;	U+1D55F	𝕟
Not;	U+02AEC	⫬
not;	U+000AC	¬
not	U+000AC	¬
NotCongruent;	U+02262	≢
NotCupCap;	U+0226D	≭
NotDoubleVerticalBar;	U+02226	∦
NotElement;	U+02209	∉
NotEqual;	U+02260	≠
NotEqualTilde;	U+02242 U+00338	≂̸
NotExists;	U+02204	∄
NotGreater;	U+0226F	≯
NotGreaterEqual;	U+02271	≱
NotGreaterFullEqual;	U+02267 U+00338	≧̸
NotGreaterGreater;	U+0226B U+00338	≫̸
NotGreaterLess;	U+02279	≹
NotGreaterSlantEqual;	U+02A7E U+00338	⩾̸
NotGreaterTilde;	U+02275	≵
NotHumpDownHump;	U+0224E U+00338	≎̸
NotHumpEqual;	U+0224F U+00338	≏̸
notin;	U+02209	∉
notindot;	U+022F5 U+00338	⋵̸
notinE;	U+022F9 U+00338	⋹̸
notinva;	U+02209	∉
notinvb;	U+022F7	⋷
notinvc;	U+022F6	⋶
NotLeftTriangle;	U+022EA	⋪
NotLeftTriangleBar;	U+029CF U+00338	⧏̸
NotLeftTriangleEqual;	U+022EC	⋬
NotLess;	U+0226E	≮
NotLessEqual;	U+02270	≰
NotLessGreater;	U+02278	≸
NotLessLess;	U+0226A U+00338	≪̸
NotLessSlantEqual;	U+02A7D U+00338	⩽̸
NotLessTilde;	U+02274	≴
NotNestedGreaterGreater;	U+02AA2 U+00338	⪢̸
NotNestedLessLess;	U+02AA1 U+00338	⪡̸
notni;	U+0220C	∌
notniva;	U+0220C	∌
notnivb;	U+022FE	⋾
notnivc;	U+022FD	⋽
NotPrecedes;	U+02280	⊀
NotPrecedesEqual;	U+02AAF U+00338	⪯̸
NotPrecedesSlantEqual;	U+022E0	⋠
NotReverseElement;	U+0220C	∌
NotRightTriangle;	U+022EB	⋫
NotRightTriangleBar;	U+029D0 U+00338	⧐̸
NotRightTriangleEqual;	U+022ED	⋭
NotSquareSubset;	U+0228F U+00338	⊏̸
NotSquareSubsetEqual;	U+022E2	⋢
NotSquareSuperset;	U+02290 U+00338	⊐̸
NotSquareSupersetEqual;	U+022E3	⋣
NotSubset;	U+02282 U+020D2	⊂⃒
NotSubsetEqual;	U+02288	⊈
NotSucceeds;	U+02281	⊁
NotSucceedsEqual;	U+02AB0 U+00338	⪰̸
NotSucceedsSlantEqual;	U+022E1	⋡
NotSucceedsTilde;	U+0227F U+00338	≿̸
NotSuperset;	U+02283 U+020D2	⊃⃒
NotSupersetEqual;	U+02289	⊉
NotTilde;	U+02241	≁
NotTildeEqual;	U+02244	≄
NotTildeFullEqual;	U+02247	≇
NotTildeTilde;	U+02249	≉
NotVerticalBar;	U+02224	∤
npar;	U+02226	∦
nparallel;	U+02226	∦
nparsl;	U+02AFD U+020E5	⫽⃥
npart;	U+02202 U+00338	∂̸
npolint;	U+02A14	⨔
npr;	U+02280	⊀
nprcue;	U+022E0	⋠
npre;	U+02AAF U+00338	⪯̸
nprec;	U+02280	⊀
npreceq;	U+02AAF U+00338	⪯̸
nrArr;	U+021CF	⇏
nrarr;	U+0219B	↛
nrarrc;	U+02933 U+00338	⤳̸
nrarrw;	U+0219D U+00338	↝̸
nRightarrow;	U+021CF	⇏
nrightarrow;	U+0219B	↛
nrtri;	U+022EB	⋫
nrtrie;	U+022ED	⋭
nsc;	U+02281	⊁
nsccue;	U+022E1	⋡
nsce;	U+02AB0 U+00338	⪰̸
Nscr;	U+1D4A9	𝒩
nscr;	U+1D4C3	𝓃
nshortmid;	U+02224	∤
nshortparallel;	U+02226	∦
nsim;	U+02241	≁
nsime;	U+02244	≄
nsimeq;	U+02244	≄
nsmid;	U+02224	∤
nspar;	U+02226	∦
nsqsube;	U+022E2	⋢
nsqsupe;	U+022E3	⋣
nsub;	U+02284	⊄
nsubE;	U+02AC5 U+00338	⫅̸
nsube;	U+02288	⊈
nsubset;	U+02282 U+020D2	⊂⃒
nsubseteq;	U+02288	⊈
nsubseteqq;	U+02AC5 U+00338	⫅̸
nsucc;	U+02281	⊁
nsucceq;	U+02AB0 U+00338	⪰̸
nsup;	U+02285	⊅
nsupE;	U+02AC6 U+00338	⫆̸
nsupe;	U+02289	⊉
nsupset;	U+02283 U+020D2	⊃⃒
nsupseteq;	U+02289	⊉
nsupseteqq;	U+02AC6 U+00338	⫆̸
ntgl;	U+02279	≹
Ntilde;	U+000D1	Ñ
Ntilde	U+000D1	Ñ
ntilde;	U+000F1	ñ
ntilde	U+000F1	ñ
ntlg;	U+02278	≸
ntriangleleft;	U+022EA	⋪
ntrianglelefteq;	U+022EC	⋬
ntriangleright;	U+022EB	⋫
ntrianglerighteq;	U+022ED	⋭
Nu;	U+0039D	Ν
nu;	U+003BD	ν
num;	U+00023	#
numero;	U+02116	№
numsp;	U+02007	
nvap;	U+0224D U+020D2	≍⃒
nVDash;	U+022AF	⊯
nVdash;	U+022AE	⊮
nvDash;	U+022AD	⊭
nvdash;	U+022AC	⊬
nvge;	U+02265 U+020D2	≥⃒
nvgt;	U+0003E U+020D2	>⃒
nvHarr;	U+02904	⤄
nvinfin;	U+029DE	⧞
nvlArr;	U+02902	⤂
nvle;	U+02264 U+020D2	≤⃒
nvlt;	U+0003C U+020D2	<⃒
nvltrie;	U+022B4 U+020D2	⊴⃒
nvrArr;	U+02903	⤃
nvrtrie;	U+022B5 U+020D2	⊵⃒
nvsim;	U+0223C U+020D2	∼⃒
nwarhk;	U+02923	⤣
nwArr;	U+021D6	⇖
nwarr;	U+02196	↖
nwarrow;	U+02196	↖
nwnear;	U+02927	⤧
Oacute;	U+000D3	Ó
Oacute	U+000D3	Ó
oacute;	U+000F3	ó
oacute	U+000F3	ó
oast;	U+0229B	⊛
ocir;	U+0229A	⊚
Ocirc;	U+000D4	Ô
Ocirc	U+000D4	Ô
ocirc;	U+000F4	ô
ocirc	U+000F4	ô
Ocy;	U+0041E	О
ocy;	U+0043E	о
odash;	U+0229D	⊝
Odblac;	U+00150	Ő
odblac;	U+00151	ő
odiv;	U+02A38	⨸
odot;	U+02299	⊙
odsold;	U+029BC	⦼
OElig;	U+00152	Œ
oelig;	U+00153	œ
ofcir;	U+029BF	⦿
Ofr;	U+1D512	𝔒
ofr;	U+1D52C	𝔬
ogon;	U+002DB	˛
Ograve;	U+000D2	Ò
Ograve	U+000D2	Ò
ograve;	U+000F2	ò
ograve	U+000F2	ò
ogt;	U+029C1	⧁

Name	Character(s)	Glyph
ohbar;	U+029B5	⦵
ohm;	U+003A9	Ω
oint;	U+0222E	∮
olarr;	U+021BA	↺
olcir;	U+029BE	⦾
olcross;	U+029BB	⦻
oline;	U+0203E	‾
olt;	U+029C0	⧀
Omacr;	U+0014C	Ō
omacr;	U+0014D	ō
Omega;	U+003A9	Ω
omega;	U+003C9	ω
Omicron;	U+0039F	Ο
omicron;	U+003BF	ο
omid;	U+029B6	⦶
ominus;	U+02296	⊖
Oopf;	U+1D546	𝕆
oopf;	U+1D560	𝕠
opar;	U+029B7	⦷
OpenCurlyDoubleQuote;	U+0201C	“
OpenCurlyQuote;	U+02018	‘
operp;	U+029B9	⦹
oplus;	U+02295	⊕
Or;	U+02A54	⩔
or;	U+02228	∨
orarr;	U+021BB	↻
ord;	U+02A5D	⩝
order;	U+02134	ℴ
orderof;	U+02134	ℴ
ordf;	U+000AA	ª
ordf	U+000AA	ª
ordm;	U+000BA	º
ordm	U+000BA	º
origof;	U+022B6	⊶
oror;	U+02A56	⩖
orslope;	U+02A57	⩗
orv;	U+02A5B	⩛
oS;	U+024C8	Ⓢ
Oscr;	U+1D4AA	𝒪
oscr;	U+02134	ℴ
Oslash;	U+000D8	Ø
Oslash	U+000D8	Ø
oslash;	U+000F8	ø
oslash	U+000F8	ø
osol;	U+02298	⊘
Otilde;	U+000D5	Õ
Otilde	U+000D5	Õ
otilde;	U+000F5	õ
otilde	U+000F5	õ
Otimes;	U+02A37	⨷
otimes;	U+02297	⊗
otimesas;	U+02A36	⨶
Ouml;	U+000D6	Ö
Ouml	U+000D6	Ö
ouml;	U+000F6	ö
ouml	U+000F6	ö
ovbar;	U+0233D	⌽
OverBar;	U+0203E	‾
OverBrace;	U+023DE	⏞
OverBracket;	U+023B4	⎴
OverParenthesis;	U+023DC	⏜
par;	U+02225	∥
para;	U+000B6	¶
para	U+000B6	¶
parallel;	U+02225	∥
parsim;	U+02AF3	⫳
parsl;	U+02AFD	⫽
part;	U+02202	∂
PartialD;	U+02202	∂
Pcy;	U+0041F	П
pcy;	U+0043F	п
percnt;	U+00025	%
period;	U+0002E	.
permil;	U+02030	‰
perp;	U+022A5	⊥
pertenk;	U+02031	‱
Pfr;	U+1D513	𝔓
pfr;	U+1D52D	𝔭
Phi;	U+003A6	Φ
phi;	U+003C6	φ
phiv;	U+003D5	ϕ
phmmat;	U+02133	ℳ
phone;	U+0260E	☎
Pi;	U+003A0	Π
pi;	U+003C0	π
pitchfork;	U+022D4	⋔
piv;	U+003D6	ϖ
planck;	U+0210F	ℏ
planckh;	U+0210E	ℎ
plankv;	U+0210F	ℏ
plus;	U+0002B	+
plusacir;	U+02A23	⨣
plusb;	U+0229E	⊞
pluscir;	U+02A22	⨢
plusdo;	U+02214	∔
plusdu;	U+02A25	⨥
pluse;	U+02A72	⩲
PlusMinus;	U+000B1	±
plusmn;	U+000B1	±
plusmn	U+000B1	±
plussim;	U+02A26	⨦
plustwo;	U+02A27	⨧
pm;	U+000B1	±
Poincareplane;	U+0210C	ℌ
pointint;	U+02A15	⨕
Popf;	U+02119	ℙ
popf;	U+1D561	𝕡
pound;	U+000A3	£
pound	U+000A3	£
Pr;	U+02ABB	⪻
pr;	U+0227A	≺
prap;	U+02AB7	⪷
prcue;	U+0227C	≼
prE;	U+02AB3	⪳
pre;	U+02AAF	⪯
prec;	U+0227A	≺
precapprox;	U+02AB7	⪷
preccurlyeq;	U+0227C	≼
Precedes;	U+0227A	≺
PrecedesEqual;	U+02AAF	⪯
PrecedesSlantEqual;	U+0227C	≼
PrecedesTilde;	U+0227E	≾
preceq;	U+02AAF	⪯
precnapprox;	U+02AB9	⪹
precneqq;	U+02AB5	⪵
precnsim;	U+022E8	⋨
precsim;	U+0227E	≾
Prime;	U+02033	″
prime;	U+02032	′
primes;	U+02119	ℙ
prnap;	U+02AB9	⪹
prnE;	U+02AB5	⪵
prnsim;	U+022E8	⋨
prod;	U+0220F	∏
Product;	U+0220F	∏
profalar;	U+0232E	⌮
profline;	U+02312	⌒
profsurf;	U+02313	⌓
prop;	U+0221D	∝
Proportion;	U+02237	∷
Proportional;	U+0221D	∝
propto;	U+0221D	∝
prsim;	U+0227E	≾
prurel;	U+022B0	⊰
Pscr;	U+1D4AB	𝒫
pscr;	U+1D4C5	𝓅
Psi;	U+003A8	Ψ
psi;	U+003C8	ψ
puncsp;	U+02008	
Qfr;	U+1D514	𝔔
qfr;	U+1D52E	𝔮
qint;	U+02A0C	⨌
Qopf;	U+0211A	ℚ
qopf;	U+1D562	𝕢
qprime;	U+02057	⁗
Qscr;	U+1D4AC	𝒬
qscr;	U+1D4C6	𝓆
quaternions;	U+0210D	ℍ
quatint;	U+02A16	⨖
quest;	U+0003F	?
questeq;	U+0225F	≟
QUOT;	U+00022	"
QUOT	U+00022	"
quot;	U+00022	"
quot	U+00022	"
rAarr;	U+021DB	⇛
race;	U+0223D U+00331	∽̱
Racute;	U+00154	Ŕ
racute;	U+00155	ŕ
radic;	U+0221A	√
raemptyv;	U+029B3	⦳
Rang;	U+027EB	⟫
rang;	U+027E9	⟩
rangd;	U+02992	⦒
range;	U+029A5	⦥
rangle;	U+027E9	⟩
raquo;	U+000BB	»
raquo	U+000BB	»
Rarr;	U+021A0	↠
rArr;	U+021D2	⇒
rarr;	U+02192	→
rarrap;	U+02975	⥵
rarrb;	U+021E5	⇥
rarrbfs;	U+02920	⤠
rarrc;	U+02933	⤳
rarrfs;	U+0291E	⤞
rarrhk;	U+021AA	↪
rarrlp;	U+021AC	↬
rarrpl;	U+02945	⥅
rarrsim;	U+02974	⥴
Rarrtl;	U+02916	⤖
rarrtl;	U+021A3	↣
rarrw;	U+0219D	↝
rAtail;	U+0291C	⤜
ratail;	U+0291A	⤚
ratio;	U+02236	∶
rationals;	U+0211A	ℚ
RBarr;	U+02910	⤐
rBarr;	U+0290F	⤏
rbarr;	U+0290D	⤍
rbbrk;	U+02773	❳
rbrace;	U+0007D	}
rbrack;	U+0005D	]
rbrke;	U+0298C	⦌
rbrksld;	U+0298E	⦎
rbrkslu;	U+02990	⦐
Rcaron;	U+00158	Ř
rcaron;	U+00159	ř
Rcedil;	U+00156	Ŗ
rcedil;	U+00157	ŗ
rceil;	U+02309	⌉
rcub;	U+0007D	}
Rcy;	U+00420	Р
rcy;	U+00440	р
rdca;	U+02937	⤷
rdldhar;	U+02969	⥩
rdquo;	U+0201D	”
rdquor;	U+0201D	”
rdsh;	U+021B3	↳
Re;	U+0211C	ℜ
real;	U+0211C	ℜ
realine;	U+0211B	ℛ
realpart;	U+0211C	ℜ
reals;	U+0211D	ℝ
rect;	U+025AD	▭
REG;	U+000AE	®
REG	U+000AE	®
reg;	U+000AE	®
reg	U+000AE	®
ReverseElement;	U+0220B	∋
ReverseEquilibrium;	U+021CB	⇋
ReverseUpEquilibrium;	U+0296F	⥯
rfisht;	U+0297D	⥽
rfloor;	U+0230B	⌋
Rfr;	U+0211C	ℜ
rfr;	U+1D52F	𝔯
rHar;	U+02964	⥤
rhard;	U+021C1	⇁
rharu;	U+021C0	⇀
rharul;	U+0296C	⥬
Rho;	U+003A1	Ρ
rho;	U+003C1	ρ
rhov;	U+003F1	ϱ
RightAngleBracket;	U+027E9	⟩
RightArrow;	U+02192	→
Rightarrow;	U+021D2	⇒
rightarrow;	U+02192	→
RightArrowBar;	U+021E5	⇥
RightArrowLeftArrow;	U+021C4	⇄
rightarrowtail;	U+021A3	↣
RightCeiling;	U+02309	⌉
RightDoubleBracket;	U+027E7	⟧
RightDownTeeVector;	U+0295D	⥝
RightDownVector;	U+021C2	⇂
RightDownVectorBar;	U+02955	⥕
RightFloor;	U+0230B	⌋
rightharpoondown;	U+021C1	⇁
rightharpoonup;	U+021C0	⇀
rightleftarrows;	U+021C4	⇄
rightleftharpoons;	U+021CC	⇌
rightrightarrows;	U+021C9	⇉
rightsquigarrow;	U+0219D	↝
RightTee;	U+022A2	⊢
RightTeeArrow;	U+021A6	↦
RightTeeVector;	U+0295B	⥛
rightthreetimes;	U+022CC	⋌
RightTriangle;	U+022B3	⊳
RightTriangleBar;	U+029D0	⧐
RightTriangleEqual;	U+022B5	⊵
RightUpDownVector;	U+0294F	⥏
RightUpTeeVector;	U+0295C	⥜
RightUpVector;	U+021BE	↾
RightUpVectorBar;	U+02954	⥔

Name	Character(s)	Glyph
RightVector;	U+021C0	⇀
RightVectorBar;	U+02953	⥓
ring;	U+002DA	˚
risingdotseq;	U+02253	≓
rlarr;	U+021C4	⇄
rlhar;	U+021CC	⇌
rlm;	U+0200F	‏
rmoust;	U+023B1	⎱
rmoustache;	U+023B1	⎱
rnmid;	U+02AEE	⫮
roang;	U+027ED	⟭
roarr;	U+021FE	⇾
robrk;	U+027E7	⟧
ropar;	U+02986	⦆
Ropf;	U+0211D	ℝ
ropf;	U+1D563	𝕣
roplus;	U+02A2E	⨮
rotimes;	U+02A35	⨵
RoundImplies;	U+02970	⥰
rpar;	U+00029	)
rpargt;	U+02994	⦔
rppolint;	U+02A12	⨒
rrarr;	U+021C9	⇉
Rrightarrow;	U+021DB	⇛
rsaquo;	U+0203A	›
Rscr;	U+0211B	ℛ
rscr;	U+1D4C7	𝓇
Rsh;	U+021B1	↱
rsh;	U+021B1	↱
rsqb;	U+0005D	]
rsquo;	U+02019	’
rsquor;	U+02019	’
rthree;	U+022CC	⋌
rtimes;	U+022CA	⋊
rtri;	U+025B9	▹
rtrie;	U+022B5	⊵
rtrif;	U+025B8	▸
rtriltri;	U+029CE	⧎
RuleDelayed;	U+029F4	⧴
ruluhar;	U+02968	⥨
rx;	U+0211E	℞
Sacute;	U+0015A	Ś
sacute;	U+0015B	ś
sbquo;	U+0201A	‚
Sc;	U+02ABC	⪼
sc;	U+0227B	≻
scap;	U+02AB8	⪸
Scaron;	U+00160	Š
scaron;	U+00161	š
sccue;	U+0227D	≽
scE;	U+02AB4	⪴
sce;	U+02AB0	⪰
Scedil;	U+0015E	Ş
scedil;	U+0015F	ş
Scirc;	U+0015C	Ŝ
scirc;	U+0015D	ŝ
scnap;	U+02ABA	⪺
scnE;	U+02AB6	⪶
scnsim;	U+022E9	⋩
scpolint;	U+02A13	⨓
scsim;	U+0227F	≿
Scy;	U+00421	С
scy;	U+00441	с
sdot;	U+022C5	⋅
sdotb;	U+022A1	⊡
sdote;	U+02A66	⩦
searhk;	U+02925	⤥
seArr;	U+021D8	⇘
searr;	U+02198	↘
searrow;	U+02198	↘
sect;	U+000A7	§
sect	U+000A7	§
semi;	U+0003B	;
seswar;	U+02929	⤩
setminus;	U+02216	∖
setmn;	U+02216	∖
sext;	U+02736	✶
Sfr;	U+1D516	𝔖
sfr;	U+1D530	𝔰
sfrown;	U+02322	⌢
sharp;	U+0266F	♯
SHCHcy;	U+00429	Щ
shchcy;	U+00449	щ
SHcy;	U+00428	Ш
shcy;	U+00448	ш
ShortDownArrow;	U+02193	↓
ShortLeftArrow;	U+02190	←
shortmid;	U+02223	∣
shortparallel;	U+02225	∥
ShortRightArrow;	U+02192	→
ShortUpArrow;	U+02191	↑

Name	Character(s)	Glyph
shy;	U+000AD	
shy	U+000AD	
Sigma;	U+003A3	Σ
sigma;	U+003C3	σ
sigmaf;	U+003C2	ς
sigmav;	U+003C2	ς
sim;	U+0223C	∼
simdot;	U+02A6A	⩪
sime;	U+02243	≃
simeq;	U+02243	≃
simg;	U+02A9E	⪞
simgE;	U+02AA0	⪠
siml;	U+02A9D	⪝
simlE;	U+02A9F	⪟
simne;	U+02246	≆
simplus;	U+02A24	⨤
simrarr;	U+02972	⥲
slarr;	U+02190	←
SmallCircle;	U+02218	∘
smallsetminus;	U+02216	∖
smashp;	U+02A33	⨳
smeparsl;	U+029E4	⧤
smid;	U+02223	∣
smile;	U+02323	⌣
smt;	U+02AAA	⪪
smte;	U+02AAC	⪬
smtes;	U+02AAC U+0FE00	⪬
SOFTcy;	U+0042C	Ь
softcy;	U+0044C	ь
sol;	U+0002F	/
solb;	U+029C4	⧄
solbar;	U+0233F	⌿
Sopf;	U+1D54A	𝕊
sopf;	U+1D564	𝕤
spades;	U+02660	♠
spadesuit;	U+02660	♠
spar;	U+02225	∥
sqcap;	U+02293	⊓
sqcaps;	U+02293 U+0FE00	⊓
sqcup;	U+02294	⊔
sqcups;	U+02294 U+0FE00	⊔
Sqrt;	U+0221A	√
sqsub;	U+0228F	⊏
sqsube;	U+02291	⊑
sqsubset;	U+0228F	⊏
sqsubseteq;	U+02291	⊑
sqsup;	U+02290	⊐
sqsupe;	U+02292	⊒
sqsupset;	U+02290	⊐
sqsupseteq;	U+02292	⊒
squ;	U+025A1	□
Square;	U+025A1	□
square;	U+025A1	□
SquareIntersection;	U+02293	⊓
SquareSubset;	U+0228F	⊏
SquareSubsetEqual;	U+02291	⊑
SquareSuperset;	U+02290	⊐
SquareSupersetEqual;	U+02292	⊒
SquareUnion;	U+02294	⊔
squarf;	U+025AA	▪
squf;	U+025AA	▪
srarr;	U+02192	→
Sscr;	U+1D4AE	𝒮
sscr;	U+1D4C8	𝓈
ssetmn;	U+02216	∖
ssmile;	U+02323	⌣
sstarf;	U+022C6	⋆
Star;	U+022C6	⋆
star;	U+02606	☆
starf;	U+02605	★
straightepsilon;	U+003F5	ϵ
straightphi;	U+003D5	ϕ
strns;	U+000AF	¯
Sub;	U+022D0	⋐
sub;	U+02282	⊂
subdot;	U+02ABD	⪽
subE;	U+02AC5	⫅
sube;	U+02286	⊆
subedot;	U+02AC3	⫃
submult;	U+02AC1	⫁
subnE;	U+02ACB	⫋
subne;	U+0228A	⊊
subplus;	U+02ABF	⪿
subrarr;	U+02979	⥹
Subset;	U+022D0	⋐
subset;	U+02282	⊂
subseteq;	U+02286	⊆
subseteqq;	U+02AC5	⫅
SubsetEqual;	U+02286	⊆
subsetneq;	U+0228A	⊊
subsetneqq;	U+02ACB	⫋

Name	Character(s)	Glyph
subsim;	U+02AC7	⫇
subsub;	U+02AD5	⫕
subsup;	U+02AD3	⫓
succ;	U+0227B	≻
succapprox;	U+02AB8	⪸
succcurlyeq;	U+0227D	≽
Succeeds;	U+0227B	≻
SucceedsEqual;	U+02AB0	⪰
SucceedsSlantEqual;	U+0227D	≽
SucceedsTilde;	U+0227F	≿
succeq;	U+02AB0	⪰
succnapprox;	U+02ABA	⪺
succneqq;	U+02AB6	⪶
succnsim;	U+022E9	⋩
succsim;	U+0227F	≿
SuchThat;	U+0220B	∋
Sum;	U+02211	∑
sum;	U+02211	∑
sung;	U+0266A	♪
Sup;	U+022D1	⋑
sup;	U+02283	⊃
sup1;	U+000B9	¹
sup1	U+000B9	¹
sup2;	U+000B2	²
sup2	U+000B2	²
sup3;	U+000B3	³
sup3	U+000B3	³
supdot;	U+02ABE	⪾
supdsub;	U+02AD8	⫘
supE;	U+02AC6	⫆
supe;	U+02287	⊇
supedot;	U+02AC4	⫄
Superset;	U+02283	⊃
SupersetEqual;	U+02287	⊇
suphsol;	U+027C9	⟉
suphsub;	U+02AD7	⫗
suplarr;	U+0297B	⥻
supmult;	U+02AC2	⫂
supnE;	U+02ACC	⫌
supne;	U+0228B	⊋
supplus;	U+02AC0	⫀
Supset;	U+022D1	⋑
supset;	U+02283	⊃
supseteq;	U+02287	⊇
supseteqq;	U+02AC6	⫆
supsetneq;	U+0228B	⊋
supsetneqq;	U+02ACC	⫌
supsim;	U+02AC8	⫈
supsub;	U+02AD4	⫔
supsup;	U+02AD6	⫖
swarhk;	U+02926	⤦
swArr;	U+021D9	⇙
swarr;	U+02199	↙
swarrow;	U+02199	↙
swnwar;	U+0292A	⤪
szlig;	U+000DF	ß
szlig	U+000DF	ß
Tab;	U+00009	␉
target;	U+02316	⌖
Tau;	U+003A4	Τ
tau;	U+003C4	τ
tbrk;	U+023B4	⎴
Tcaron;	U+00164	Ť
tcaron;	U+00165	ť
Tcedil;	U+00162	Ţ
tcedil;	U+00163	ţ
Tcy;	U+00422	Т
tcy;	U+00442	т
tdot;	U+020DB	⃛
telrec;	U+02315	⌕
Tfr;	U+1D517	𝔗
tfr;	U+1D531	𝔱
there4;	U+02234	∴
Therefore;	U+02234	∴
therefore;	U+02234	∴
Theta;	U+00398	Θ
theta;	U+003B8	θ
thetasym;	U+003D1	ϑ
thetav;	U+003D1	ϑ
thickapprox;	U+02248	≈
thicksim;	U+0223C	∼
ThickSpace;	U+0205F U+0200A	
thinsp;	U+02009	
ThinSpace;	U+02009	
thkap;	U+02248	≈
thksim;	U+0223C	∼
THORN;	U+000DE	Þ
THORN	U+000DE	Þ
thorn;	U+000FE	þ
thorn	U+000FE	þ
Tilde;	U+0223C	∼

Name	Character(s)	Glyph
tilde;	U+002DC	˜
TildeEqual;	U+02243	≃
TildeFullEqual;	U+02245	≅
TildeTilde;	U+02248	≈
times;	U+000D7	×
times	U+000D7	×
timesb;	U+022A0	⊠
timesbar;	U+02A31	⨱
timesd;	U+02A30	⨰
tint;	U+0222D	∭
toea;	U+02928	⤨
top;	U+022A4	⊤
topbot;	U+02336	⌶
topcir;	U+02AF1	⫱
Topf;	U+1D54B	𝕋
topf;	U+1D565	𝕥
topfork;	U+02ADA	⫚
tosa;	U+02929	⤩
tprime;	U+02034	‴
TRADE;	U+02122	™
trade;	U+02122	™
triangle;	U+025B5	▵
triangledown;	U+025BF	▿
triangleleft;	U+025C3	◃
trianglelefteq;	U+022B4	⊴
triangleq;	U+0225C	≜
triangleright;	U+025B9	▹
trianglerighteq;	U+022B5	⊵
tridot;	U+025EC	◬
trie;	U+0225C	≜
triminus;	U+02A3A	⨺
TripleDot;	U+020DB	◌⃛
triplus;	U+02A39	⨹
trisb;	U+029CD	⧍
tritime;	U+02A3B	⨻
trpezium;	U+023E2	⏢
Tscr;	U+1D4AF	𝒯
tscr;	U+1D4C9	𝓉
TScy;	U+00426	Ц
tscy;	U+00446	ц
TSHcy;	U+0040B	Ћ
tshcy;	U+0045B	ћ
Tstrok;	U+00166	Ŧ
tstrok;	U+00167	ŧ
twixt;	U+0226C	≬
twoheadleftarrow;	U+0219E	↞
twoheadrightarrow;	U+021A0	↠
Uacute;	U+000DA	Ú
Uacute	U+000DA	Ú
uacute;	U+000FA	ú
uacute	U+000FA	ú
Uarr;	U+0219F	↟
uArr;	U+021D1	⇑
uarr;	U+02191	↑
Uarrocir;	U+02949	⥉
Ubrcy;	U+0040E	Ў
ubrcy;	U+0045E	ў
Ubreve;	U+0016C	Ŭ
ubreve;	U+0016D	ŭ
Ucirc;	U+000DB	Û
Ucirc	U+000DB	Û
ucirc;	U+000FB	û
ucirc	U+000FB	û
Ucy;	U+00423	У
ucy;	U+00443	у
udarr;	U+021C5	⇅
Udblac;	U+00170	Ű
udblac;	U+00171	ű
udhar;	U+0296E	⥮
ufisht;	U+0297E	⥾
Ufr;	U+1D518	𝔘
ufr;	U+1D532	𝔲
Ugrave;	U+000D9	Ù
Ugrave	U+000D9	Ù
ugrave;	U+000F9	ù
ugrave	U+000F9	ù
uHar;	U+02963	⥣
uharl;	U+021BF	↿
uharr;	U+021BE	↾
uhblk;	U+02580	▀
ulcorn;	U+0231C	⌜
ulcorner;	U+0231C	⌜
ulcrop;	U+0230F	⌏
ultri;	U+025F8	◸
Umacr;	U+0016A	Ū
umacr;	U+0016B	ū
uml;	U+000A8	¨
uml	U+000A8	¨
UnderBar;	U+0005F	_
UnderBrace;	U+023DF	⏟
UnderBracket;	U+023B5	⎵

Name	Character(s)	Glyph
UnderParenthesis;	U+023DD	⏝
Union;	U+022C3	⋃
UnionPlus;	U+0228E	⊎
Uogon;	U+00172	Ų
uogon;	U+00173	ų
Uopf;	U+1D54C	𝕌
uopf;	U+1D566	𝕦
UpArrow;	U+02191	↑
Uparrow;	U+021D1	⇑
uparrow;	U+02191	↑
UpArrowBar;	U+02912	⤒
UpArrowDownArrow;	U+021C5	⇅
UpDownArrow;	U+02195	↕
Updownarrow;	U+021D5	⇕
updownarrow;	U+02195	↕
UpEquilibrium;	U+0296E	⥮
upharpoonleft;	U+021BF	↿
upharpoonright;	U+021BE	↾
uplus;	U+0228E	⊎
UpperLeftArrow;	U+02196	↖
UpperRightArrow;	U+02197	↗
Upsi;	U+003D2	ϒ
upsi;	U+003C5	υ
upsih;	U+003D2	ϒ
Upsilon;	U+003A5	Υ
upsilon;	U+003C5	υ
UpTee;	U+022A5	⊥
UpTeeArrow;	U+021A5	↥
upuparrows;	U+021C8	⇈
urcorn;	U+0231D	⌝
urcorner;	U+0231D	⌝
urcrop;	U+0230E	⌎
Uring;	U+0016E	Ů
uring;	U+0016F	ů
urtri;	U+025F9	◹
Uscr;	U+1D4B0	𝒰
uscr;	U+1D4CA	𝓊
utdot;	U+022F0	⋰
Utilde;	U+00168	Ũ
utilde;	U+00169	ũ
utri;	U+025B5	▵
utrif;	U+025B4	▴
uuarr;	U+021C8	⇈
Uuml;	U+000DC	Ü
Uuml	U+000DC	Ü
uuml;	U+000FC	ü
uuml	U+000FC	ü
uwangle;	U+029A7	⦧
vangrt;	U+0299C	⦜
varepsilon;	U+003F5	ϵ
varkappa;	U+003F0	ϰ
varnothing;	U+02205	∅
varphi;	U+003D5	ϕ
varpi;	U+003D6	ϖ
varpropto;	U+0221D	∝
vArr;	U+021D5	⇕
varr;	U+02195	↕
varrho;	U+003F1	ϱ
varsigma;	U+003C2	ς
varsubsetneq;	U+0228A U+0FE00	⊊
varsubsetneqq;	U+02ACB U+0FE00	⫋
varsupsetneq;	U+0228B U+0FE00	⊋
varsupsetneqq;	U+02ACC U+0FE00	⫌
vartheta;	U+003D1	ϑ
vartriangleleft;	U+022B2	⊲
vartriangleright;	U+022B3	⊳
Vbar;	U+02AEB	⫫
vBar;	U+02AE8	⫨
vBarv;	U+02AE9	⫩
Vcy;	U+00412	В
vcy;	U+00432	в
VDash;	U+022AB	⊫
Vdash;	U+022A9	⊩
vDash;	U+022A8	⊨
vdash;	U+022A2	⊢
Vdashl;	U+02AE6	⫦
Vee;	U+022C1	⋁
vee;	U+02228	∨
veebar;	U+022BB	⊻
veeeq;	U+0225A	≚
vellip;	U+022EE	⋮
Verbar;	U+02016	‖
verbar;	U+0007C	|
Vert;	U+02016	‖
vert;	U+0007C	|
VerticalBar;	U+02223	|
VerticalLine;	U+0007C	|
VerticalSeparator;	U+02758	❘
VerticalTilde;	U+02240	≀
VeryThinSpace;	U+0200A	
Vfr;	U+1D519	𝔙

Name	Character(s)	Glyph
vfr;	U+1D533	𝔳
vltri;	U+022B2	⊲
vnsub;	U+02282 U+020D2	⊂⃒
vnsup;	U+02283 U+020D2	⊃⃒
Vopf;	U+1D54D	𝕍
vopf;	U+1D567	𝕧
vprop;	U+0221D	∝
vrtri;	U+022B3	⊳
Vscr;	U+1D4B1	𝒱
vscr;	U+1D4CB	𝓋
vsubnE;	U+02ACB U+0FE00	⫋
vsubne;	U+0228A U+0FE00	⊊
vsupnE;	U+02ACC U+0FE00	⫌
vsupne;	U+0228B U+0FE00	⊋
Vvdash;	U+022AA	⊪
vzigzag;	U+0299A	⦚
Wcirc;	U+00174	Ŵ
wcirc;	U+00175	ŵ
wedbar;	U+02A5F	⩟
Wedge;	U+022C0	⋀
wedge;	U+02227	∧
wedgeq;	U+02259	≙
weierp;	U+02118	℘
Wfr;	U+1D51A	𝔚
wfr;	U+1D534	𝔴
Wopf;	U+1D54E	𝕎
wopf;	U+1D568	𝕨
wp;	U+02118	℘
wr;	U+02240	≀
wreath;	U+02240	≀
Wscr;	U+1D4B2	𝒲
wscr;	U+1D4CC	𝓌
xcap;	U+022C2	⋂
xcirc;	U+025EF	◯
xcup;	U+022C3	⋃
xdtri;	U+025BD	▽
Xfr;	U+1D51B	𝔛
xfr;	U+1D535	𝔵
xhArr;	U+027FA	⟺
xharr;	U+027F7	⟷
Xi;	U+0039E	Ξ
xi;	U+003BE	ξ
xlArr;	U+027F8	⟸
xlarr;	U+027F5	⟵
xmap;	U+027FC	⟼
xnis;	U+022FB	⋻
xodot;	U+02A00	⨀
Xopf;	U+1D54F	𝕏
xopf;	U+1D569	𝕩
xoplus;	U+02A01	⨁
xotime;	U+02A02	⨂
xrArr;	U+027F9	⟹
xrarr;	U+027F6	⟶
Xscr;	U+1D4B3	𝒳
xscr;	U+1D4CD	𝓍
xsqcup;	U+02A06	⨆
xuplus;	U+02A04	⨄
xutri;	U+025B3	△
xvee;	U+022C1	⋁
xwedge;	U+022C0	⋀
Yacute;	U+000DD	Ý
Yacute	U+000DD	Ý
yacute;	U+000FD	ý
yacute	U+000FD	ý
YAcy;	U+0042F	Я
yacy;	U+0044F	я
Ycirc;	U+00176	Ŷ
ycirc;	U+00177	ŷ
Ycy;	U+0042B	Ы
ycy;	U+0044B	ы
yen;	U+000A5	¥
yen	U+000A5	¥
Yfr;	U+1D51C	𝔜
yfr;	U+1D536	𝔶
YIcy;	U+00407	Ї
yicy;	U+00457	ї
Yopf;	U+1D550	𝕐
yopf;	U+1D56A	𝕪
Yscr;	U+1D4B4	𝒴
yscr;	U+1D4CE	𝓎
YUcy;	U+0042E	Ю
yucy;	U+0044E	ю
Yuml;	U+00178	Ÿ
yuml;	U+000FF	ÿ
yuml	U+000FF	ÿ
Zacute;	U+00179	Ź
zacute;	U+0017A	ź
Zcaron;	U+0017D	Ž
zcaron;	U+0017E	ž
Zcy;	U+00417	З
zcy;	U+00437	з

Name	Character(s)	Glyph
Zdot;	U+0017B	Ż
zdot;	U+0017C	ż
zeetrf;	U+02128	ℨ
ZeroWidthSpace;	U+0200B	
Zeta;	U+00396	Ζ
zeta;	U+003B6	ζ

Name	Character(s)	Glyph
Zfr;	U+02128	ℨ
zfr;	U+1D537	𝔷
ZHcy;	U+00416	Ж
zhcy;	U+00436	ж
zigrarr;	U+021DD	⇝
Zopf;	U+02124	ℤ

Name	Character(s)	Glyph
zopf;	U+1D56B	𝕫
Zscr;	U+1D4B5	𝒵
zscr;	U+1D4CF	𝓏
zwj;	U+0200D	
zwnj;	U+0200C	

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]*[p1588]

Note

This list is static and will not be expanded or changed in the future.

14 The XML syntax §p14 84

Note

This section only describes the rules for XML resources. Rules for `text/html`p1546 resources are discussed in the section above entitled "The HTML syntaxp1356".

⚠Warning!

Using the XML syntax is not recommended, for reasons which include the fact that there is no specification which defines the rules for how an XML parser must map a string of bytes or characters into a `Document`p135 object, as well as the fact that the XML syntax is essentially unmaintained — in that, it's not expected that any further features will ever be added to the XML syntax (even when such features have been added to the HTML syntaxp1356).

14.1 Writing documents in the XML syntax §p14 84

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]p1588 [XMLNS]p1588

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 `<`, `>`, `&`, `"`, and `'`).*

14.2 Parsing XML documents §p14 84

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`p135 object.

Note

At the time of writing, no such rules actually exist.

An XML parserp1484 is either associated with a `Document`p135 object when it is created, or creates one implicitly.

This `Document`p135 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*. These nodes must be created using the following algorithms:

- For elements, create an element for a tokenp1418, given the element's prefix, to ensure the proper element interfaces are created and that custom elementsp796 are set up correctly.
- For `Text` nodes, create a text node.
- For `CDATASection` nodes, create a CDATA section node.

- For `Comment` nodes, create a comment node.
- For `ProcessingInstruction` nodes, create a processing instruction node.
- For `DocumentType` nodes, create a doctype.

For the operations that the XML parser[p1484] performs on the `Document`[p135]'s tree, 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. [XML][p1588] [XMLNS][p1588] [DOM][p1582] [UIEVENTS][p1587]

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[p1383].

Note

This is used by various elements to only start certain processes once they are popped off of the stack of open elements[p1383].

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[p1474] section.) [XML][p1588]

- `-//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`
- `-//WAPFORUM//DTD XHTML Mobile 1.1//EN`
- `-//WAPFORUM//DTD XHTML Mobile 1.2//EN`

Furthermore, user agents should attempt to retrieve the above external entity's content when one of the above public identifiers is used, and should not attempt to retrieve any other external entity's content.

Note

This is not strictly a violation of XML*, but it does contradict the spirit of* XML*'s requirements. This is motivated by a desire for user agents to all handle entities in an interoperable fashion without requiring any network access for handling external subsets. [XML][p1588]*

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[p1485].

When an XML parser[p1484] with XML scripting support enabled[p1485] creates a `script`[p685] element, it must have its parser document[p692] set and its force async[p692] set to false. If the parser was created as part of the XML fragment parsing algorithm[p1487], then the element's already started[p692] must be set to true. When the element's end tag is subsequently parsed, the user agent must perform a microtask checkpoint[p1200], and then prepare[p694] the `script`[p685] element. If this causes there to be a pending parsing-blocking script[p698], then the user agent must run the following steps:

1. Block this instance of the XML parser[p1484], such that the event loop[p1193] will not run tasks[p1194] that invoke it.
2. Spin the event loop[p1201] until the parser's `Document`[p135] has no style sheet that is blocking scripts[p212] and the pending parsing-blocking script[p698]'s ready to be parser-executed[p692] is true.
3. Unblock this instance of the XML parser[p1484], such that tasks[p1194] that invoke it can again be run.
4. Execute the script element[p698] given by the pending parsing-blocking script[p698].
5. Set the pending parsing-blocking script[p698] to null.

Note

Since the `document.write()`[p1223] API is not available for XML documents, much of the complexity in the HTML parser[p1368] is not needed in the XML parser[p1484].

Note

When the XML parser[p1484] has XML scripting support disabled[p1485], none of this happens.

When an XML parser[p1484] would append a node to a `template`[p705] element, it must instead append it to the `template`[p705] element's template contents[p707] (a `DocumentFragment` node).

Note

This is a willful violation of XML*; unfortunately, XML is not formally extensible in the manner that is needed for* `template`[p705] *processing. [XML][p1588]*

When an XML parser[p1484] 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 at a time. In such cases, the XML parser[p1484] must act as it would have if faced with a single string consisting of the concatenation of all those characters.

When an XML parser[p1484] reaches the end of its input, it must stop parsing[p1458], following the same rules as the HTML parser[p1368]. An XML parser[p1484] can also be aborted[p1459], which must again be done in the same way as for an HTML parser[p1368].

For the purposes of conformance checkers, if a resource is determined to be in the XML syntax[p1484], then it is an XML document.

14.3 Serializing XML fragments §[p1486]

The **XML fragment serialization algorithm** for a `Document`[p135] or `Element` node either returns a fragment of XML that represents that node or throws an exception.

For `Document`[p135]s, the algorithm must return a string in the form of a document entity, if none of the error cases below apply.

For `Element`s, 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[p1486], 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`[p705] elements

The relevant child nodes[p1486] are the child nodes of the `template`[p705] element's template contents[p707], if any.

For all other nodes

The relevant child nodes[p1486] are the child nodes of node itself, if any.

For `Element`s, 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`[p135] case.) [XML][p1588] [XMLNS][p1588]

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`[p135] 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][p1588]
- 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][p1588]

- 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][p1588]
- An Attr node with no namespace whose local name is the lowercase string "xmlns". [XMLNS][p1588]
- 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][p1588]
- A Comment node whose data contains two adjacent U+002D HYPHEN-MINUS characters (-) or ends with such a character.
- A ProcessingInstruction node whose target name is an ASCII case-insensitive match for the string "xml".
- A ProcessingInstruction node whose target name contains a U+003A COLON (:).
- A ProcessingInstruction node whose data contains the string "?>".

Note

These are the only ways to make a DOM unserialisable. The DOM enforces all the other XML constraints; for example, trying to append two elements to a Document[p135] node will throw a "HierarchyRequestError" DOMException.

14.4 Parsing XML fragments §[p1487]

The **XML fragment parsing algorithm** given an Element or DocumentFragment node *target* and a string *input*, runs the following steps. They return a DocumentFragment.

1. Let *context*[p1473] be *target* if *target* is an Element; otherwise *target*'s host.
2. Assert: *context*[p1473] is non-null.
3. Create a new XML parser[p1484].
4. Feed the parser[p1486] just created the string corresponding to the start tag of *context*[p1473], 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.
5. Feed the parser[p1486] just created the string *input*.
6. Feed the parser[p1486] just created the string corresponding to the end tag of *context*[p1473].
7. If there is an XML well-formedness or XML namespace well-formedness error, then throw a "SyntaxError" DOMException.
8. If the document element of the resulting Document[p135] has any sibling nodes, then throw a "SyntaxError" DOMException.
9. Let *newChildren* be the resulting Document[p135] node's document element's children, in tree order.
10. Let *fragment* be the result of creating a document fragment given *context*'s node document.
11. For each *node* of *newChildren*, in tree order: append *node* to *fragment*.
12. Return *fragment*.

15 Rendering §[p14] 88

*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[p49], the term "**expected**" in this section has the same conformance implications as "must".*

15.1 Introduction §[p14] 88

The suggestions in this section are generally expressed in CSS terms. User agents are expected[p1488] to either support CSS, or translate from the CSS rules given in this section to approximations for other presentation mechanisms.

In the absence of style-layer rules to the contrary (e.g. author style sheets), user agents are expected[p1488] to render an element so that it conveys to the user the meaning that the element represents[p148], 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[p1488]. The presence of the `hidden`[p862] attribute normally means the element is not being rendered[p1488], though this might be overridden by the style sheets.

Note

The fully active[p1051] state does not affect whether an element is being rendered[p1488] or not. Even if a document is not fully active[p1051] and not shown at all to the user, elements within it can still qualify as "being rendered".

An element is said to **intersect the viewport** when it is being rendered[p1488] and its associated CSS layout box intersects the viewport.

Note

Similar to the being rendered[p1488] state, elements in non-fully active[p1051] documents can still intersect the viewport[p1488]. The viewport is not shared between documents and might not always be shown to the user, so an element in a non-fully active[p1051] document can still intersect the viewport associated with its document.

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. [INTERSECTIONOBSERVER][p1583]

An element is **delegating its rendering to its children** if it is not being rendered[p1488] but its children (if any) could be rendered[p1488], as a result of CSS 'display: contents', or some equivalent in other styling languages. [CSSDISPLAY][p1580]

User agents that do not honor author-level CSS style sheets are nonetheless expected[p1488] to act as if they applied the CSS rules given in these sections in a manner consistent with this specification and the relevant CSS and Unicode specifications. [CSS][p1580] [UNICODE][p1587] [BIDI][p1579]

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 §p14 89

The CSS rules given in these subsections are, except where otherwise specified, expected[p1488] to be used as part of the user-agent level style sheet defaults for all documents that contain HTML elements[p46].

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[p81] doesn't generate an error, then the user agent is expected[p1488] to use the parsed value as a pixel length for a presentational hint[p1489] 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[p83] doesn't generate an error, then the user agent is expected[p1488] to use the parsed dimension as the value for a presentational hint[p1489] 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[p84] doesn't generate an error, then the user agent is expected[p1488] to use the parsed dimension as the value for a presentational hint[p1489] 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 a pair of attributes *w* and *h* on an element *element* **map to the aspect-ratio property**, it means that if *element* has both attributes *w* and *h*, and parsing those attributes' values using the rules for parsing non-negative integers[p81] doesn't generate an error for either, then the user agent is expected[p1488] to use the parsed integers as a presentational hint[p1489] for the 'aspect-ratio' property of the form `auto w / h`.

When the text below says that a pair of attributes *w* and *h* on an element *element* **map to the aspect-ratio property (using dimension rules)**, it means that if *element* has both attributes *w* and *h*, and parsing those attributes' values using the rules for parsing dimension values[p83] doesn't generate an error or return a percentage for either, then the user agent is expected[p1488] to use the parsed dimensions as a presentational hint[p1489] for the 'aspect-ratio' property of the form `auto w / h`.

When a user agent is to **align descendants** of a node, the user agent is expected[p1488] 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[p1489] a particular descendant, the most deeply nested such element is expected[p1488] to override the others. Aligned elements are expected[p1488] to be aligned by having the used values of their margins on the line-left and line-right sides be set accordingly. [CSSLOGICAL][p1581] [CSSWM][p1582]

15.3 Non-replaced elements §p14 89

15.3.1 Hidden elements §p14 89

CSS

```css
@namespace "http://www.w3.org/1999/xhtml";

area, base, basefont, datalist, head, link, meta, noembed,
noframes, param, rp, script, style, template, title {
  display: none;
}
```

```
[hidden]:not([hidden=until-found i]):not(embed) {
  display: none;
}

[hidden=until-found i]:not(embed) {
  content-visibility: hidden;
}

embed[hidden] { display: inline; height: 0; width: 0; }

input[type=hidden i] { display: none !important; }

@media (scripting) {
  noscript { display: none !important; }
}
```

15.3.2 The page §[p14] 90

CSS
```
@namespace "http://www.w3.org/1999/xhtml";

html, body { display: block; }
```

For each property in the table below, given a body[p212] element, the first attribute that exists maps to the pixel length property[p1489] on the body[p212] 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[p1488] to be used for that property instead.

Property	Source
'margin-top', 'margin-bottom'	The body[p212] element's marginheight[p1534] attribute
	The body[p212] element's topmargin[p1534] attribute
	The body[p212] element's container frame element[p1490]'s marginheight[p1534] attribute
'margin-left', 'margin-right'	The body[p212] element's marginwidth[p1534] attribute
	The body[p212] element's leftmargin[p1534] attribute
	The body[p212] element's container frame element[p1490]'s marginwidth[p1534] attribute

If the body[p212] element's node document's node navigable[p1036] is a child navigable[p1039], and the container[p1038] of that navigable[p1035] is a frame[p1537] or iframe[p404] element, then the **container frame element** of the body[p212] element is that frame[p1537] or iframe[p404] element. Otherwise, there is no container frame element[p1490].

⚠Warning!

The above requirements imply that a page can change the margins of another page (including one from another origin[p939]) using, for example, an iframe[p404]. This is potentially a security risk, as it might in some cases allow an attack to contrive a situation in which a page is rendered not as the author intended, possibly for the purposes of phishing or otherwise misleading the user.

If a Document[p135]'s node navigable[p1036] is a child navigable[p1039], then it is expected[p1488] to be positioned and sized to fit inside the content box of the container[p1038] of that navigable[p1035]. If the container[p1038] is not being rendered[p1488], the navigable[p1035] is expected[p1488] to have a viewport with zero width and zero height.

If a Document[p135]'s node navigable[p1036] is a child navigable[p1039], the container[p1038] of that navigable[p1035] is a frame[p1537] or iframe[p404] element, that element has a `scrolling` attribute, and that attribute's value is an ASCII case-insensitive match for the string "`off`", "`noscroll`", or "`no`", then the user agent is expected[p1488] to prevent any scrollbars from being shown for the viewport of the Document[p135]'s node navigable[p1036], regardless of the 'overflow' property that applies to that viewport.

When a body[p212] element has a background[p1535] attribute set to a non-empty value, the new value is expected[p1488] to be encoding-parsed-and-serialized[p101] relative to the element's node document, and if that does not return failure, the user agent is expected[p1488] to

treat the attribute as a presentational hint[p1489] setting the element's 'background-image' property to the return value.

When a body[p212] element has a bgcolor[p1534] attribute set, the new value is expected[p1488] to be parsed using the rules for parsing a legacy color value[p98], and if that does not return failure, the user agent is expected[p1488] to treat the attribute as a presentational hint[p1489] setting the element's 'background-color' property to the resulting color.

When a body[p212] element has a text[p1534] attribute, its value is expected[p1488] to be parsed using the rules for parsing a legacy color value[p98], and if that does not return failure, the user agent is expected[p1488] to treat the attribute as a presentational hint[p1489] setting the element's 'color' property to the resulting color.

When a body[p212] element has a link[p1534] attribute, its value is expected[p1488] to be parsed using the rules for parsing a legacy color value[p98], and if that does not return failure, the user agent is expected[p1488] to treat the attribute as a presentational hint[p1489] setting the 'color' property of any element in the Document[p135] matching the :link[p821] pseudo-class to the resulting color.

When a body[p212] element has a vlink[p1534] attribute, its value is expected[p1488] to be parsed using the rules for parsing a legacy color value[p98], and if that does not return failure, the user agent is expected[p1488] to treat the attribute as a presentational hint[p1489] setting the 'color' property of any element in the Document[p135] matching the :visited[p821] pseudo-class to the resulting color.

When a body[p212] element has an alink[p1534] attribute, its value is expected[p1488] to be parsed using the rules for parsing a legacy color value[p98], and if that does not return failure, the user agent is expected[p1488] to treat the attribute as a presentational hint[p1489] setting the 'color' property of any element in the Document[p135] matching the :active[p821] pseudo-class and either the :link[p821] pseudo-class or the :visited[p821] pseudo-class to the resulting color.

15.3.3 Flow content §[p1491]

CSS

```css
@namespace "http://www.w3.org/1999/xhtml";

address, blockquote, center, dialog, div, figure, figcaption, footer, form,
header, hr, legend, listing, main, p, plaintext, pre, search, xmp {
  display: block;
}

blockquote, figure, listing, p, plaintext, pre, xmp {
  margin-block: 1em;
}

blockquote, figure { margin-inline: 40px; }

address { font-style: italic; }
listing, plaintext, pre, xmp {
  font-family: monospace; white-space: pre;
}

dialog:not([open]) { display: none; }
dialog {
  position: absolute;
  inset-inline-start: 0; inset-inline-end: 0;
  width: fit-content;
  height: fit-content;
  margin: auto;
  border: solid;
  padding: 1em;
  background-color: Canvas;
  color: CanvasText;
}
dialog:modal {
  position: fixed;
  overflow: auto;
  inset-block: 0;
  max-width: calc(100% - 6px - 2em);
```

```
  max-height: calc(100% - 6px - 2em);
}
dialog::backdrop {
  background: rgba(0,0,0,0.1);
}

[popover]:not(:popover-open):not(dialog[open]) {
  display:none;
}

dialog:popover-open {
  display:block;
}

[popover] {
  position: fixed;
  inset: 0;
  width: fit-content;
  height: fit-content;
  margin: auto;
  border: solid;
  padding: 0.25em;
  overflow: auto;
  color: CanvasText;
  background-color: Canvas;
}

:popover-open::backdrop {
  position: fixed;
  inset: 0;
  pointer-events: none !important;
  background-color: transparent;
}

slot {
  display: contents;
}
```

The following rules are also expected[p1488] to apply, as presentational hints[p1489]:

CSS
```
@namespace "http://www.w3.org/1999/xhtml";

pre[wrap] { white-space: pre-wrap; }
```

In quirks mode, the following rules are also expected[p1488] to apply:

CSS
```
@namespace "http://www.w3.org/1999/xhtml";

form { margin-block-end: 1em; }
```

The center[p1531] element, and the div[p266] element when it has an align[p1534] attribute whose value is an ASCII case-insensitive match for either the string "center" or the string "middle", are expected[p1488] to center text within themselves, as if they had their 'text-align' property set to 'center' in a presentational hint[p1489], and to align descendants[p1489] to the center.

The div[p266] element, when it has an align[p1534] attribute whose value is an ASCII case-insensitive match for the string "left", is expected[p1488] to left-align text within itself, as if it had its 'text-align' property set to 'left' in a presentational hint[p1489], and to align descendants[p1489] to the left.

The div[p266] element, when it has an align[p1534] attribute whose value is an ASCII case-insensitive match for the string "right", is expected[p1488] to right-align text within itself, as if it had its 'text-align' property set to 'right' in a presentational hint[p1489], and to align

descendants[p1489] to the right.

The div[p266] element, when it has an align[p1534] attribute whose value is an ASCII case-insensitive match for the string "justify", is expected[p1488] to full-justify text within itself, as if it had its 'text-align' property set to 'justify' in a presentational hint[p1489], and to align descendants[p1489] to the left.

15.3.4 Phrasing content §[p1493]

CSS
```
@namespace "http://www.w3.org/1999/xhtml";

cite, dfn, em, i, var { font-style: italic; }
b, strong { font-weight: bolder; }
code, kbd, samp, tt { font-family: monospace; }
big { font-size: larger; }
small { font-size: smaller; }

sub { vertical-align: sub; }
sup { vertical-align: super; }
sub, sup { line-height: normal; font-size: smaller; }

ruby { display: ruby; }
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 */
nobr { white-space: nowrap; }
wbr { display-outside: break-opportunity; } /* this also has bidi implications */
nobr wbr { white-space: normal; }
```

The following rules are also expected[p1488] to apply, as presentational hints[p1489]:

CSS
```
@namespace "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 ruby[p281] elements that are not rt[p287] or rp[p288] elements are expected[p1488] to be wrapped in anonymous boxes whose 'display' property has the value 'ruby-base'. [CSSRUBY][p1581]

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

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 ruby elements or multiple sequential 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 rt elements in the absence of rp elements.

User agents are expected to support the 'clear' property on inline elements (in order to render br elements with clear attributes) in the manner described in the non-normative note to this effect in *CSS*.

The initial value for the 'color' property is expected to be black. The initial value for the 'background-color' property is expected to be 'transparent'. The canvas's background is expected to be white.

When a font 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 failure, 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 font element has a `face` attribute, the user agent is expected to treat the attribute as a presentational hint setting the element's 'font-family' property to the attribute's value.

When a font element has a `size` attribute, the user agent is expected to use the following steps, known as the **rules for parsing a legacy font size**, to treat the attribute as a presentational hint setting the element's 'font-size' property:

1. Let *input* be the attribute's value.
2. Let *position* be a pointer into *input*, initially pointing at the start of the string.
3. Skip ASCII whitespace within *input* given *position*.
4. If *position* is past the end of *input*, there is no presentational hint. Return.
5. If the character at *position* is a U+002B PLUS SIGN character (+), then let *mode* be *relative-plus*, and advance *position* to the next character. Otherwise, if the character at *position* is a U+002D HYPHEN-MINUS character (-), then let *mode* be *relative-minus*, and advance *position* to the next character. Otherwise, let *mode* be *absolute*.
6. Collect a sequence of code points that are ASCII digits from *input* given *position*, and let *digits* be the resulting sequence.
7. If *digits* is the empty string, there is no presentational hint. Return.
8. Interpret *digits* as a base-ten integer. Let *value* be the resulting number.
9. If *mode* is *relative-plus*, then increment *value* by 3. If *mode* is *relative-minus*, then let *value* be the result of subtracting *value* from 3.
10. If *value* is greater than 7, let it be 7.
11. If *value* is less than 1, let it be 1.
12. Set the 'font-size' property to the keyword corresponding to the value of *value* according to the following table:

value	'font-size' keyword
1	'x-small'
2	'small'
3	'medium'
4	'large'
5	'x-large'
6	'xx-large'
7	'xxx-large'

15.3.5 Bidirectional text §[p1495]

CSS
```
@namespace "http://www.w3.org/1999/xhtml";

[dir]:dir(ltr), bdi:dir(ltr), input[type=tel i]:dir(ltr) { direction: ltr; }
[dir]:dir(rtl), bdi:dir(rtl) { direction: rtl; }

address, blockquote, center, div, figure, figcaption, footer, form, header, hr,
legend, listing, main, p, plaintext, pre, summary, xmp, article, aside,
:heading, hgroup, nav, search, section, table, caption, colgroup, col, thead,
tbody, tfoot, tr, td, th, dir, dd, dl, dt, menu, ol, ul, li, bdi, output,
[dir=ltr i], [dir=rtl i], [dir=auto i] {
  unicode-bidi: isolate;
}

bdo, bdo[dir] { unicode-bidi: isolate-override; }

input[dir=auto i]: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 */
```

When an input[p539] element's dir[p167] attribute is in the auto[p167] state and its type[p541] attribute is in the Text[p546] state, then the user agent is expected[p1488] 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[p606] elements, and input[p539] elements when their type[p541] attribute is in the Text[p546], Search[p546], Telephone[p547], URL[p547], or Email[p548] state) are expected[p1488] 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[p1488] to apply, following those above: [ENCODING][p1582]

CSS
```
@namespace "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,
:heading, hgroup, nav, search, section, table, caption, colgroup, col, thead,
tbody, tfoot, tr, td, th, dir, dd, dl, dt, menu, ol, ul, li, [dir=ltr i],
[dir=rtl i], [dir=auto i], *|* {
  unicode-bidi: bidi-override;
}
input:not([type=submit i]):not([type=reset i]):not([type=button i]),
textarea {
  unicode-bidi: normal;
}
```

15.3.6 Sections and headings §[p1495]

CSS
```
@namespace "http://www.w3.org/1999/xhtml";

article, aside, :heading, hgroup, nav, section {
  display: block;
}

:heading { font-weight: bold; }
```

```
:heading(1) { margin-block: 0.67em; font-size: 2.00em; }
:heading(2) { margin-block: 0.83em; font-size: 1.50em; }
:heading(3) { margin-block: 1.00em; font-size: 1.17em; }
:heading(4) { margin-block: 1.33em; font-size: 1.00em; }
:heading(5) { margin-block: 1.67em; font-size: 0.83em; }
:heading(6, 7, 8, 9) {
  font-size: 0.67em;
  margin-block: 2.33em;
}
```

15.3.7 Lists §[p1496]

CSS
```
@namespace "http://www.w3.org/1999/xhtml";

dir, dd, dl, dt, menu, ol, ul { display: block; }
li { display: list-item; text-align: match-parent; }

dir, dl, menu, ol, ul { margin-block: 1em; }

:is(dir, dl, menu, ol, ul) :is(dir, dl, menu, ol, ul) {
  margin-block: 0;
}

dd { margin-inline-start: 40px; }
dir, menu, ol, ul { padding-inline-start: 40px; }

ol, ul, menu { counter-reset: list-item; }
ol { list-style-type: decimal; }

dir, menu, ul {
  list-style-type: disc;
}
:is(dir, menu, ol, ul) :is(dir, menu, ul) {
  list-style-type: circle;
}
:is(dir, menu, ol, ul) :is(dir, menu, ol, ul) :is(dir, menu, ul) {
  list-style-type: square;
}
```

The following rules are also expected[p1488] to apply, as presentational hints[p1489]:

CSS
```
@namespace "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=i s], li[type=i 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; }
```

In quirks mode, the following rules are also expected[p1488] to apply:

CSS
```
@namespace "http://www.w3.org/1999/xhtml";

li { list-style-position: inside; }
```

```
li :is(dir, menu, ol, ul) { list-style-position: outside; }
:is(dir, menu, ol, ul) :is(dir, menu, ol, ul, li) { list-style-position: unset; }
```

When rendering li[p251] elements, non-CSS user agents are expected[p1488] to use the ordinal value[p252] of the li[p251] element to render the counter in the list item marker.

For CSS user agents, some aspects of rendering list items are defined by the *CSS Lists* specification. Additionally, the following attribute mappings are expected[p1488] to apply: [CSSLISTS][p1581]

When an li[p251] element has a value[p251] attribute, and parsing that attribute's value using the rules for parsing integers[p80] doesn't generate an error, the user agent is expected[p1488] to use the parsed value *value* as a presentational hint[p1489] for the 'counter-set' property of the form `list-item` *value*.

When an ol[p247] element has a start[p248] attribute or a reversed[p248] attribute, or both, the user agent is expected[p1488] to use the following steps to treat the attributes as a presentational hint[p1489] for the 'counter-reset' property:

1. Let *value* be null.
2. If the element has a start[p248] attribute, then set *value* to the result of parsing the attribute's value using the rules for parsing integers[p80].
3. If the element has a reversed[p248] attribute:
    1. If *value* is an integer, then increment *value* by 1 and return `reversed(list-item)` *value*.
    2. Otherwise, return `reversed(list-item)`.

    Note

    Either the start[p248] attribute was absent, or parsing its value resulted in an error.
4. Otherwise:
    1. If *value* is an integer, then decrement *value* by 1 and return `list-item` *value*.
    2. Otherwise, there is no presentational hint[p1489].

15.3.8 Tables §[p1497]

CSS

```
@namespace "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 {
  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 i], table[rules=groups i], table[rules=rows i],
table[rules=cols i], table[rules=all i], table[frame=void i],
table[frame=above i], table[frame=below i], table[frame=hsides i],
table[frame=lhs i], table[frame=rhs i], table[frame=vsides i],
table[frame=box i], table[frame=border i],
table[rules=none i] > tr > td, table[rules=none i] > tr > th,
table[rules=groups i] > tr > td, table[rules=groups i] > tr > th,
table[rules=rows i] > tr > td, table[rules=rows i] > tr > th,
table[rules=cols i] > tr > td, table[rules=cols i] > tr > th,
table[rules=all i] > tr > td, table[rules=all i] > tr > th,
table[rules=none i] > thead > tr > td, table[rules=none i] > thead > tr > th,
table[rules=groups i] > thead > tr > td, table[rules=groups i] > thead > tr > th,
table[rules=rows i] > thead > tr > td, table[rules=rows i] > thead > tr > th,
table[rules=cols i] > thead > tr > td, table[rules=cols i] > thead > tr > th,
table[rules=all i] > thead > tr > td, table[rules=all i] > thead > tr > th,
table[rules=none i] > tbody > tr > td, table[rules=none i] > tbody > tr > th,
table[rules=groups i] > tbody > tr > td, table[rules=groups i] > tbody > tr > th,
table[rules=rows i] > tbody > tr > td, table[rules=rows i] > tbody > tr > th,
table[rules=cols i] > tbody > tr > td, table[rules=cols i] > tbody > tr > th,
table[rules=all i] > tbody > tr > td, table[rules=all i] > tbody > tr > th,
table[rules=none i] > tfoot > tr > td, table[rules=none i] > tfoot > tr > th,
table[rules=groups i] > tfoot > tr > td, table[rules=groups i] > tfoot > tr > th,
table[rules=rows i] > tfoot > tr > td, table[rules=rows i] > tfoot > tr > th,
table[rules=cols i] > tfoot > tr > td, table[rules=cols i] > tfoot > tr > th,
table[rules=all i] > tfoot > tr > td, table[rules=all i] > tfoot > tr > th {
  border-color: black;
}
```

The following rules are also expected[p1488] to apply, as presentational hints[p1489]:

CSS

```
@namespace "http://www.w3.org/1999/xhtml";

table[align=left i] { float: left; }
table[align=right i] { float: right; }
table[align=center i] { margin-inline: auto; }
thead[align=absmiddle i], tbody[align=absmiddle i], tfoot[align=absmiddle i],
tr[align=absmiddle i], td[align=absmiddle i], th[align=absmiddle i] {
  text-align: center;
}

caption[align=bottom i] { caption-side: bottom; }
p[align=left i], h1[align=left i], h2[align=left i], h3[align=left i],
h4[align=left i], h5[align=left i], h6[align=left i] {
  text-align: left;
}
p[align=right i], h1[align=right i], h2[align=right i], h3[align=right i],
h4[align=right i], h5[align=right i], h6[align=right i] {
  text-align: right;
}
p[align=center i], h1[align=center i], h2[align=center i], h3[align=center i],
h4[align=center i], h5[align=center i], h6[align=center i] {
  text-align: center;
}
p[align=justify i], h1[align=justify i], h2[align=justify i], h3[align=justify i],
```

```
h4[align=justify i], h5[align=justify i], h6[align=justify i] {
  text-align: justify;
}
thead[valign=top i], tbody[valign=top i], tfoot[valign=top i],
tr[valign=top i], td[valign=top i], th[valign=top i] {
  vertical-align: top;
}
thead[valign=middle i], tbody[valign=middle i], tfoot[valign=middle i],
tr[valign=middle i], td[valign=middle i], th[valign=middle i] {
  vertical-align: middle;
}
thead[valign=bottom i], tbody[valign=bottom i], tfoot[valign=bottom i],
tr[valign=bottom i], td[valign=bottom i], th[valign=bottom i] {
  vertical-align: bottom;
}
thead[valign=baseline i], tbody[valign=baseline i], tfoot[valign=baseline i],
tr[valign=baseline i], td[valign=baseline i], th[valign=baseline i] {
  vertical-align: baseline;
}

td[nowrap], th[nowrap] { white-space: nowrap; }

table[rules=none i], table[rules=groups i], table[rules=rows i],
table[rules=cols i], table[rules=all i] {
  border-style: hidden;
  border-collapse: collapse;
}
table[border] { border-style: outset; } /* only if border is not equivalent to zero */
table[frame=void i] { border-style: hidden; }
table[frame=above i] { border-style: outset hidden hidden hidden; }
table[frame=below i] { border-style: hidden hidden outset hidden; }
table[frame=hsides i] { border-style: outset hidden outset hidden; }
table[frame=lhs i] { border-style: hidden hidden hidden outset; }
table[frame=rhs i] { border-style: hidden outset hidden hidden; }
table[frame=vsides i] { border-style: hidden outset; }
table[frame=box i], table[frame=border i] { border-style: outset; }

table[border] > tr > td, table[border] > tr > th,
table[border] > thead > tr > td, table[border] > thead > tr > th,
table[border] > tbody > tr > td, table[border] > tbody > tr > th,
table[border] > tfoot > tr > td, table[border] > tfoot > tr > th {
  /* only if border is not equivalent to zero */
  border-width: 1px;
  border-style: inset;
}
table[rules=none i] > tr > td, table[rules=none i] > tr > th,
table[rules=none i] > thead > tr > td, table[rules=none i] > thead > tr > th,
table[rules=none i] > tbody > tr > td, table[rules=none i] > tbody > tr > th,
table[rules=none i] > tfoot > tr > td, table[rules=none i] > tfoot > tr > th,
table[rules=groups i] > tr > td, table[rules=groups i] > tr > th,
table[rules=groups i] > thead > tr > td, table[rules=groups i] > thead > tr > th,
table[rules=groups i] > tbody > tr > td, table[rules=groups i] > tbody > tr > th,
table[rules=groups i] > tfoot > tr > td, table[rules=groups i] > tfoot > tr > th,
table[rules=rows i] > tr > td, table[rules=rows i] > tr > th,
table[rules=rows i] > thead > tr > td, table[rules=rows i] > thead > tr > th,
table[rules=rows i] > tbody > tr > td, table[rules=rows i] > tbody > tr > th,
table[rules=rows i] > tfoot > tr > td, table[rules=rows i] > tfoot > tr > th {
  border-width: 1px;
  border-style: none;
}
table[rules=cols i] > tr > td, table[rules=cols i] > tr > th,
table[rules=cols i] > thead > tr > td, table[rules=cols i] > thead > tr > th,
```

```
table[rules=cols i] > tbody > tr > td, table[rules=cols i] > tbody > tr > th,
table[rules=cols i] > tfoot > tr > td, table[rules=cols i] > tfoot > tr > th {
  border-width: 1px;
  border-block-style: none;
  border-inline-style: solid;
}
table[rules=all i] > tr > td, table[rules=all i] > tr > th,
table[rules=all i] > thead > tr > td, table[rules=all i] > thead > tr > th,
table[rules=all i] > tbody > tr > td, table[rules=all i] > tbody > tr > th,
table[rules=all i] > tfoot > tr > td, table[rules=all i] > tfoot > tr > th {
  border-width: 1px;
  border-style: solid;
}

table[rules=groups i] > colgroup {
  border-inline-width: 1px;
  border-inline-style: solid;
}
table[rules=groups i] > thead,
table[rules=groups i] > tbody,
table[rules=groups i] > tfoot {
  border-block-width: 1px;
  border-block-style: solid;
}

table[rules=rows i] > tr, table[rules=rows i] > thead > tr,
table[rules=rows i] > tbody > tr, table[rules=rows i] > tfoot > tr {
  border-block-width: 1px;
  border-block-style: solid;
}
```

In quirks mode, the following rules are also expected[p1488] to apply:

CSS
```
@namespace "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[p506] element is expected[p1488] to be treated as if it was present as many times as its span[p507] attribute specifies[p81].

For the purposes of the CSS table model, the colgroup[p505] element, if it contains no col[p506] element, is expected[p1488] to be treated as if it had as many such children as its span[p506] attribute specifies[p81].

For the purposes of the CSS table model, the colspan[p515] and rowspan[p515] attributes on td[p512] and th[p513] elements are expected[p1488] to provide[p81] the *special knowledge* regarding cells spanning rows and columns.

In HTML documents, the following rules are also expected[p1488] to apply:

CSS
```
@namespace "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 'border-spacing' on the element.

The table element's cellpadding attribute maps to the pixel length properties 'padding-top', 'padding-right', 'padding-bottom', and 'padding-left' of any td and th elements that have corresponding cells in the table corresponding to the table element.

The table element's height attribute maps to the dimension property 'height' on the table element.

The table element's width attribute maps to the dimension property (ignoring zero) 'width' on the table element.

The col element's width attribute maps to the dimension property 'width' on the col element.

The thead, tbody, and tfoot elements' height attribute maps to the dimension property 'height' on the element.

The tr element's height attribute maps to the dimension property 'height' on the tr element.

The td and th elements' height attributes map to the dimension property (ignoring zero) 'height' on the element.

The td and th elements' width attributes map to the dimension property (ignoring zero) 'width' on the element.

The thead, tbody, tfoot, tr, td, and th elements, when they have an `align` attribute whose value is an ASCII case-insensitive match for either the string "`center`" or the string "`middle`", are expected to center text within themselves, as if they had their 'text-align' property set to 'center' in a presentational hint, and to align descendants to the center.

The thead, tbody, tfoot, tr, td, and th elements, when they have an `align` attribute whose value is an ASCII case-insensitive match for the string "`left`", are expected to left-align text within themselves, as if they had their 'text-align' property set to 'left' in a presentational hint, and to align descendants to the left.

The thead, tbody, tfoot, tr, td, and th elements, when they have an `align` attribute whose value is an ASCII case-insensitive match for the string "`right`", are expected to right-align text within themselves, as if they had their 'text-align' property set to 'right' in a presentational hint, and to align descendants to the right.

The thead, tbody, tfoot, tr, td, and th elements, when they have an `align` attribute whose value is an ASCII case-insensitive match for the string "`justify`", are expected to full-justify text within themselves, as if they had their 'text-align' property set to 'justify' in a presentational hint, and to align descendants to the left.

User agents are expected to have a rule in their user agent 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 encoding-parsed-and-serialized relative to the element's node document, and if that does not return failure, the user agent is expected to treat the attribute as a presentational hint setting the element's 'background-image' property to the return value.

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 failure, the user agent is expected to treat the attribute as a presentational hint setting the element's 'background-color' property to the resulting color.

When a table element has a bordercolor attribute, its value is expected to be parsed using the rules for parsing a legacy color value, and if that does not return failure, the user agent is expected to treat the attribute as a presentational hint setting the element's 'border-top-color', 'border-right-color', 'border-bottom-color', and 'border-left-color' properties to the resulting color.

The table element's border attribute maps to the pixel length properties 'border-top-width', 'border-right-width', 'border-bottom-width', 'border-left-width' on the element. If the attribute is present but parsing the attribute's value using the rules for parsing non-negative integers generates an error, a default value of 1px is expected to be used for that property instead.

Rules marked "**only if border is not equivalent to zero**" in the CSS block above is expected to only be applied if the border attribute mentioned in the selectors for the rule is not only present but, when parsed using the rules for parsing non-negative integers, is also found to have a value other than zero or to generate an error.

In quirks mode, a td element or a th element that has a nowrap attribute but also has a width attribute whose value, when parsed using the rules for parsing 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

```css
@namespace "http://www.w3.org/1999/xhtml";

input, button, textarea {
  letter-spacing: initial;
  word-spacing: initial;
  line-height: initial;
}

input, select, button, textarea {
  text-transform: initial;
  text-indent: initial;
  text-shadow: initial;
  appearance: auto;
}

input:not([type=range i], [type=checkbox i], [type=radio i]) {
  overflow: clip !important;
  overflow-clip-margin: 0 !important;
}

input[type=image i] {
  overflow-clip-margin: content-box !important;
}

input, select, textarea {
  text-align: initial;
}
```

```
:autofill {
  field-sizing: fixed !important;
}

input:is([type=reset i], [type=button i], [type=submit i]), button {
  text-align: center;
}

input, button {
  display: inline-block;
}

input[type=hidden i], input[type=file i], input[type=image i] {
  appearance: none;
}

input:is([type=radio i], [type=checkbox i], [type=reset i], [type=button i],
[type=submit i], [type=color i], [type=search i]), select, button {
  box-sizing: border-box;
}

textarea { white-space: pre-wrap; }

input:not([type=file i], [type=range i], [type=checkbox i], [type=radio i]) {
  align-content: unsafe center;
}

button,
::file-selector-button,
input:is([type=color i], [type=reset i], [type=button i], [type=submit i]) {
  align-content: center;
}
```

In quirks mode, the following rules are also expected[p1488] to apply:

CSS
```
@namespace "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[p1511] section, which describes the look and feel of the control.

For input[p539] elements where the type[p541] attribute is not in the Hidden[p546] state or the Image Button[p566] state, and that are being rendered[p1488], are expected[p1488] to act as follows:

- The inner display type is always 'flow-root'.

15.3.11 The hr[p241] element §[p1503]

CSS
```
@namespace "http://www.w3.org/1999/xhtml";

hr {
  color: gray;
  border-style: inset;
  border-width: 1px;
  margin-block: 0.5em;
  margin-inline: auto;
  overflow: hidden;
}
```

The following rules are also expected[p1488] to apply, as presentational hints[p1489]:

```css
@namespace "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[p241] element has either a color[p1534] attribute or a noshade[p1534] attribute, and furthermore also has a size[p1534] attribute, and parsing that attribute's value using the rules for parsing non-negative integers[p81] doesn't generate an error, then the user agent is expected[p1488] to use the parsed value divided by two as a pixel length for presentational hints[p1489] for the properties 'border-top-width', 'border-right-width', 'border-bottom-width', and 'border-left-width' on the element.

Otherwise, if an hr[p241] element has neither a color[p1534] attribute nor a noshade[p1534] attribute, but does have a size[p1534] attribute, and parsing that attribute's value using the rules for parsing non-negative integers[p81] doesn't generate an error, then: if the parsed value is one, then the user agent is expected[p1488] to use the attribute as a presentational hint[p1489] setting the element's 'border-bottom-width' to 0; otherwise, if the parsed value is greater than one, then the user agent is expected[p1488] to use the parsed value minus two as a pixel length for presentational hints[p1489] for the 'height' property on the element.

The width[p1534] attribute on an hr[p241] element maps to the dimension property[p1489] 'width' on the element.

When an hr[p241] element has a color[p1534] attribute, its value is expected[p1488] to be parsed using the rules for parsing a legacy color value[p98], and if that does not return failure, the user agent is expected[p1488] to treat the attribute as a presentational hint[p1489] setting the element's 'color' property to the resulting color.

15.3.12 The fieldset[p620] and legend[p623] elements §[p1504]

```css
@namespace "http://www.w3.org/1999/xhtml";

fieldset {
  display: block;
  margin-inline: 2px;
  border: groove 2px ThreeDFace;
  padding-block: 0.35em 0.625em;
  padding-inline: 0.75em;
  min-inline-size: min-content;
}

legend {
  padding-inline: 2px;
}

legend[align=left i] {
  justify-self: left;
}

legend[align=center i] {
  justify-self: center;
}

legend[align=right i] {
  justify-self: right;
}
```

The fieldset[p620] element, when it generates a CSS box, is expected[p1488] to act as follows:

- The element is expected[p1488] to establish a new block formatting context.
- The 'display' property is expected[p1488] 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`[p623] 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[p1505], then the border is expected[p1488] 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[p1505]'s margin rectangle at its static position (ignoring transforms), and the block-start outer edge of the `fieldset`[p620]'s border.
  2. The block-end edge of the rectangle is the larger of the block-end edge of the rendered legend[p1505]'s margin rectangle at its static position (ignoring transforms), and the block-end outer edge of the `fieldset`[p620]'s border.
  3. The inline-start edge of the rectangle is the smaller of the inline-start edge of the rendered legend[p1505]'s border rectangle at its static position (ignoring transforms), and the inline-start outer edge of the `fieldset`[p620]'s border.
  4. The inline-end edge of the rectangle is the larger of the inline-end edge of the rendered legend[p1505]'s border rectangle at its static position (ignoring transforms), and the inline-end outer edge of the `fieldset`[p620]'s border.
- The space allocated for the element's border on the block-start side is expected[p1488] to be the element's 'border-block-start-width' or the rendered legend[p1505]'s margin box size in the `fieldset`[p620]'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[p1505]'s margin box that spills out past the border, if any, is expected[p1488] to be subtracted from the 'block-size'. If the content box's block-size would be negative, then let the content box's block-size be 0 instead.
- If the element has a rendered legend[p1505], then that element is expected[p1488] to be the first child box.
- The anonymous fieldset content box[p1506] is expected[p1488] to appear after the rendered legend[p1505] and is expected[p1488] to contain the content (including the '::before' and '::after' pseudo-elements) of the `fieldset`[p620] element except for the rendered legend[p1505], if there is one.
- The used value of the 'padding-top', 'padding-right', 'padding-bottom', and 'padding-left' properties are expected[p1488] 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[p1505] and the min-content inline size of the anonymous fieldset content box[p1506].
- For the purpose of calculating the max-content inline size, use the greater of the max-content inline size of the rendered legend[p1505] and the max-content inline size of the anonymous fieldset content box[p1506].

A `fieldset`[p620] element's rendered legend[p1505], if any, is expected[p1488] to act as follows:

- The element is expected[p1488] 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[p1488] to behave as if its computed value was blockified.

Note

This does not change the computed value.

- If the computed value of 'inline-size' is 'auto', then the used value is the fit-content inline size.
- The element is expected[p1488] 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[p1488] to be constrained in the inline direction by the inline content size of the `fieldset`[p620] as if

it had used its computed inline padding.

Example

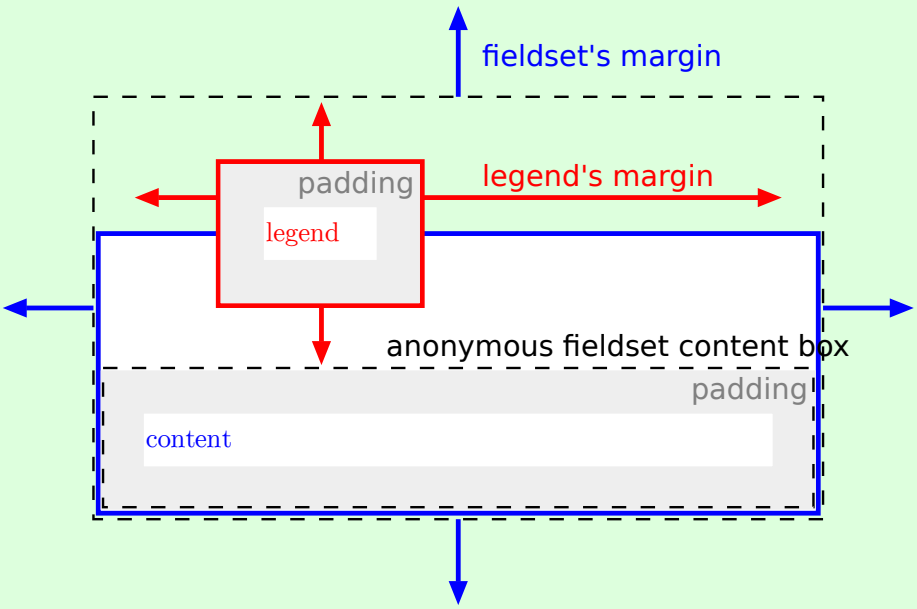
For example, if the `fieldset` has a specified padding of 50px, then the rendered legend will be positioned 50px in from the `fieldset`'s border. The padding will further apply to the anonymous fieldset content box instead of the `fieldset` element itself.

- 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.

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'.
  - 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-width'
  - 'flex-direction'
  - 'flex-wrap'
  - 'gap'
  - 'grid-auto-columns'
  - 'grid-auto-flow'
  - 'grid-auto-rows'
  - 'grid-template-areas'
  - 'grid-template-columns'
  - 'grid-template-rows'
  - 'justify-content'
  - 'justify-items'
  - 'overflow'
  - 'padding-bottom'
  - 'padding-left'
  - 'padding-right'
  - 'padding-top'
  - 'rule'
  - 'rule-break'
  - 'rule-inset'
  - 'rule-overlap'
  - 'rule-visibility-items'
  - '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



The legend is rendered over the top border, and the top border area reserves vertical space for the legend. The fieldset's top margin starts at the top margin edge of the legend. The legend's horizontal margins, or the 'justify-self' property, gives its horizontal position. The anonymous fieldset content box[p1506] appears below the legend.

15.4 Replaced elements §[p1507]

Note

The following elements can be replaced elements: audio[p425], canvas[p713], embed[p413], iframe[p404], img[p359], input[p539], object[p416], and video[p420].

15.4.1 Embedded content §[p1507]

The embed[p413], iframe[p404], and video[p420] elements are expected[p1488] to be treated as replaced elements.

A canvas[p713] element that represents[p148] embedded content[p158] is expected[p1488] to be treated as a replaced element; the contents of such elements are the element's bitmap, if any, or else a transparent black bitmap with the same natural dimensions as the element. Other canvas[p713] elements are expected[p1488] to be treated as ordinary elements in the rendering model.

An object[p416] element that represents[p148] an image, plugin, or its content navigable[p1038] is expected[p1488] to be treated as a replaced element. Other object[p416] elements are expected[p1488] to be treated as ordinary elements in the rendering model.

The audio[p425] element, when it is exposing a user interface[p481], is expected[p1488] 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[p425] element is not exposing a user interface[p481], the user agent is expected[p1488] to force its 'display' property to compute to 'none', irrespective of CSS rules.

Whether a video[p420] element is exposing a user interface[p481] is not expected[p1488] to affect the size of the rendering; controls are expected[p1488] to be overlaid above the page content without causing any layout changes, and are expected[p1488] to disappear when the user does not need them.

When a video[p420] element represents a poster frame or frame of video, the poster frame or frame of video is expected[p1488] 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[p420] element itself, and is expected[p1488] to be centered in the video[p420] element.

Any subtitles or captions are expected[p1488] to be overlaid directly on top of their video[p420] element, as defined by the relevant rendering rules; for WebVTT, those are the rules for updating the display of WebVTT text tracks. [WEBVTT][p1588]

When the user agent starts exposing a user interface[p481] for a video[p420] element, the user agent should run the rules for updating the text track rendering[p467] of each of the text tracks[p466] in the video[p420] element's list of text tracks[p466] that are showing[p467] and whose text track kind[p466] is one of subtitles[p466] or captions[p466] (e.g., for text tracks[p466] based on WebVTT, the rules for updating the display of WebVTT text tracks). [WEBVTT][p1588]

Note

Resizing video[p420] and canvas[p713] elements does not interrupt video playback or clear the canvas.

The following CSS rules are expected[p1488] to apply:

CSS

```
@namespace "http://www.w3.org/1999/xhtml";

iframe { border: 2px inset; }
video { object-fit: contain; }
```

15.4.2 Images §[p1508]

User agents are expected[p1488] to render img[p359] elements and input[p539] elements whose type[p541] attributes are in the Image Button[p566] state, according to the first applicable rules from the following list:

↪ **If the element represents[p148] an image**

The user agent is expected[p1488] 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[p148] an image and either:**

- **the user agent has reason to believe that the image will become *available*[p377] and be rendered in due course, or**
- **the element has no `alt` attribute, or**
- **the Document[p135] is in quirks mode, and the element already has natural dimensions (e.g., from the dimension attributes[p495] or CSS rules)**

The user agent is expected[p1488] 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[p539] elements, the element is expected[p1488] to appear button-like to indicate that the element is a button[p532].

↪ **If the element is an img[p359] element that represents[p148] some text and the user agent does not expect this to change**

The user agent is expected[p1488] to treat the element as a 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[p359] element that represents[p148] nothing and the user agent does not expect this to change**

The user agent is expected[p1488] to treat the element as a replaced element whose natural 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[p539] element that does not represent[p148] an image and the user agent does not expect this to change**

The user agent is expected[p1488] to treat the element as a replaced element consisting of a button whose content is the element's alternative text. The natural dimensions of the button are expected[p1488] 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[p1488] 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[p1488] 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[p1488] 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:

CSS
```
@namespace "http://www.w3.org/1999/xhtml";

img:is([sizes="auto" i], [sizes^="auto," i]) {
  contain: size !important;
  contain-intrinsic-size: 300px 150px;
}
```

The following CSS rules are expected to apply when the Document is in quirks mode:

CSS
```
@namespace "http://www.w3.org/1999/xhtml";

img[align=left i] { margin-right: 3px; }
img[align=right i] { 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 "http://www.w3.org/1999/xhtml";

embed[align=left i], iframe[align=left i], img[align=left i],
input[type=image i][align=left i], object[align=left i] {
  float: left;
}

embed[align=right i], iframe[align=right i], img[align=right i],
input[type=image i][align=right i], object[align=right i] {
  float: right;
}

embed[align=top i], iframe[align=top i], img[align=top i],
input[type=image i][align=top i], object[align=top i] {
  vertical-align: top;
}

embed[align=baseline i], iframe[align=baseline i], img[align=baseline i],
input[type=image i][align=baseline i], object[align=baseline i] {
  vertical-align: baseline;
}

embed[align=texttop i], iframe[align=texttop i], img[align=texttop i],
input[type=image i][align=texttop i], object[align=texttop i] {
  vertical-align: text-top;
}

embed[align=absmiddle i], iframe[align=absmiddle i], img[align=absmiddle i],
input[type=image i][align=absmiddle i], object[align=absmiddle i],
```

```
embed[align=abscenter i], iframe[align=abscenter i], img[align=abscenter i],
input[type=image i][align=abscenter i], object[align=abscenter i] {
  vertical-align: middle;
}

embed[align=bottom i], iframe[align=bottom i], img[align=bottom i],
input[type=image i][align=bottom i], object[align=bottom i] {
  vertical-align: bottom;
}
```

When an 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 iframe element has a frameborder attribute whose value, when parsed using the rules for parsing integers, is zero or an error, the user agent is expected to have presentational hints setting the element's 'border-top-width', 'border-right-width', 'border-bottom-width', and 'border-left-width' properties to zero.

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 an img element's dimension attribute source map to the dimension properties 'width' and 'height' on the img element respectively. They similarly map to the aspect-ratio property (using dimension rules) of the img element.

The width and height attributes on embed, iframe, object, and 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 width and height attributes map to the aspect-ratio property (using dimension rules) on img and video elements, and input elements with a type attribute in the Image Button state.

The width and height attributes map to the aspect-ratio property on canvas elements.

15.4.4 Image maps §10

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 §p15 11

15.5.1 Native appearance §p15 11

The *CSS Basic User Interface* specification calls elements that can have a native appearance widgets, and defines whether to use that native appearance depending on the 'appearance' property. That logic, in turn, depends on whether each the element is classified as a devolvable widget or non-devolvable widget. This section defines which elements match these concepts for HTML, what their native appearance is, and any particularity of their devolved state or primitive appearance. [CSSUI]p1582

The following elements can have a native appearance for the purpose of the CSS 'appearance' property.

- button p585
- input p539
- meter p616
- progress p614
- select p590
- textarea p606

15.5.2 Writing mode §p15 11

Several widgets have their rendering controlled by the 'writing-mode' CSS property. For the purposes of those widgets, we have the following definitions.

A **horizontal writing mode** is when resolving the 'writing-mode' property of the control results in a computed value of 'horizontal-tb'.

A **vertical writing mode** is when resolving the 'writing-mode' property of the control results in a computed value of either 'vertical-rl', 'vertical-lr', 'sideways-rl' or 'sideways-lr'.

15.5.3 Button layout §p15 11

When an element uses button layoutp1511, it is a devolvable widget, and its native appearance is that of a button.

Button layout is as follows:

- If the element is a button p585 element, then the 'display' property is expected p1488 to act as follows:
    - If the computed value of 'display' is 'inline-grid', 'grid', 'inline-flex', 'flex', 'none', or 'contents', 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 p1488 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. [CSS]p1580
- 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.

Need to define the primitive appearance.

15.5.4 The button p585 element §p15 11

The button p585 element, when it generates a CSS box, is expected p1488 to depict a button and to use button layout p1511.

15.5.5 The details[p666] and summary[p672] elements §[p15] 12

```css
@namespace "http://www.w3.org/1999/xhtml";

details, summary {
  display: block;
}
details > summary:first-of-type {
  display: list-item;
  counter-increment: list-item 0;
  list-style: disclosure-closed inside;
}
details[open] > summary:first-of-type {
  list-style-type: disclosure-open;
}
```

The details[p666] element is expected[p1488] to have an internal shadow tree with three child elements:

1. The first child element is a slot[p712] that is expected[p1488] to take the details[p666] element's first summary[p672] element child, if any. This element has a single child summary[p672] element called the **default summary** which has text content that is implementation-defined (and probably locale-specific).

   The summary[p672] element that this slot represents[p148] is expected[p1488] to allow the user to request the details be shown or hidden.

2. The second child element is a slot[p712] that is expected[p1488] to take the details[p666] element's remaining descendants, if any. This element has no contents.

   This element is expected[p1488] to match the '::details-content' pseudo-element.

   This element is expected[p1488] to have its style[p171] attribute set to "`display: block; content-visibility: hidden;`" when the details[p666] element does not have an open[p667] attribute. When it does have the open[p667] attribute, the style[p171] attribute is expected[p1488] to be set to "`display: block;`".

   Note

   Because the slots are hidden inside a shadow tree, this style[p171] attribute is not directly visible to author code. Its impacts, however, are visible. Notably, the choice of `content-visibility: hidden` *instead of, e.g.,* `display: none`*, impacts the results of various APIs that query layout information.*

3. The third child element is either a link[p184] or style[p207] element with the following styles for the default summary[p1512]:

   ```css
   :host summary {
     display: list-item;
     counter-increment: list-item 0;
     list-style: disclosure-closed inside;
   }
   :host([open]) summary {
     list-style-type: disclosure-open;
   }
   ```

   Note

   The position of this child element relative to the other two is not observable. This means that implementations might have it in a different order relative to its siblings. Implementations might even associate the style with the shadow tree using a mechanism that is not an element.

Note

The structure of this shadow tree is observable through the ways that the children of the details[p666] element and the '::details-content' pseudo-element respond to CSS styles.

15.5.6 The input element as a text entry widget

An input element whose type attribute is in the Text, Telephone, URL, or Email state, is a devolvable widget. Its native appearance is expected to render as an 'inline-block' box depicting a one-line text control.

An input element whose type attribute is in the Search state is a devolvable widget. Its native appearance is expected to render as an 'inline-block' box depicting a one-line text control. If the computed value of the element's 'appearance' property is not 'textfield', it may have a distinct style indicating that it is a search field.

An input element whose type attribute is in the Password state is a devolvable widget. Its native appearance is expected to render as an 'inline-block' box depicting a one-line text control that obscures data entry.

For input elements whose type attribute is in one of the above states, the used value of the 'line-height' property must be a length value that is no smaller than what the used value would be for 'line-height: normal'.

Note

The used value will not be the actual keyword 'normal'. Also, this rule does not affect the computed value.

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.

An input element whose type attribute is in one of the above states is an element with default preferred size, and user agents are expected to apply the 'field-sizing' CSS property to the element. User agents are expected to determine the inline size of its intrinsic size by the following steps:

1. If the 'field-sizing' property on the element has a computed value of 'content', the inline size is determined by the text which the element shows. The text is either a value or a short hint specified by the placeholder attribute. User agents may take the text caret size into account in the inline size.
2. If the element has a size attribute, and parsing that attribute's value using the rules for parsing non-negative integers doesn't generate an error, return the value obtained from applying the converting a character width to pixels algorithm to the value of the attribute.
3. Otherwise, return 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)×*avg* + *max*, where *size* is the character width to convert, *avg* is the average character width of the primary font for the element for which the algorithm is being run, in pixels, and *max* is the maximum character width of that same font, also in pixels. (The element's 'letter-spacing' property does not affect the result.)

These text controls are expected to be scroll containers and support scrolling in the inline axis, but not the block axis.

Need to detail the native appearance and primitive appearance.

15.5.7 The input element as domain-specific widgets

An input element whose type attribute is in the Date state is a devolvable widget expected to render as an 'inline-block' box depicting a date control.

An input element whose type attribute is in the Month state is a devolvable widget expected to render as an 'inline-block' box depicting a month control.

An input element whose type attribute is in the Week state is a devolvable widget expected to render as an 'inline-block' box depicting a week control.

An input element whose type attribute is in the Time state is a devolvable widget 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 a devolvable widget 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 a devolvable widget expected to render as an 'inline-block' box depicting a number control.

An input element whose type attribute is in the Number state is an element with default preferred size, and user agents are expected to apply the 'field-sizing' CSS property to the element. The block size of the intrinsic size is about one line high. If the 'field-sizing' property on the element has a computed value of 'content', the inline size of the intrinsic size is expected to be about as wide as necessary to show the current value. Otherwise, the inline size of the intrinsic size is expected to be about as wide as necessary to show the widest possible value.

An input element whose type attribute is in the Date, Month, Week, Time, or Local Date and Time state, is expected to be about one line high, and about as wide as necessary to show the widest possible value.

Need to detail the native appearance and primitive appearance.

15.5.8 The input element as a range control §14

An input element whose type attribute is in the Range state is a non-devolvable widget. Its native appearance is expected to render as an 'inline-block' box depicting a slider control.

When this control has a horizontal writing mode, the control is expected to be a horizontal slider. Its lowest value is on the right if the 'direction' property has a computed value of 'rtl', and on the left otherwise. When this control has a vertical writing mode, it is expected to be a vertical slider. Its lowest value is on the bottom if the 'direction' property has a computed value of 'rtl', and on the top otherwise.

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.

Need to detail the primitive appearance.

15.5.9 The input element as a color well §14

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 a single block-level block container child box, which 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.

Need to detail the native appearance and primitive appearance.

15.5.10 The input element as a checkbox and radio button widgets §14

An input element whose type attribute is in the Checkbox state is a non-devolvable widget expected to render as an 'inline-block' box containing a single checkbox control, with no label.

Need to detail the native appearance and primitive appearance.

An input element whose type attribute is in the Radio Button state is a non-devolvable widget expected to render as an 'inline-block' box containing a single radio button control, with no label.

Need to detail the native appearance and primitive appearance.

15.5.11 The input element as a file upload control §15

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 filename(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. Its contents are expected to be implementation-defined (and possibly locale-specific) text, for example "Choose file".

User agents may handle an input element whose type attribute is in the File Upload state as an element with default preferred size, and user agents may apply the 'field-sizing' CSS property to the element. If the 'field-sizing' property on the element has a computed value of 'content', the intrinsic size of the element is expected to depend on its content such as the '::file-selector-button' pseudo-element and chosen file names.

15.5.12 The input element as a button §15

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 its contents 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.

15.5.13 The marquee element §15

CSS
```
@namespace "http://www.w3.org/1999/xhtml";

marquee {
  display: inline-block;
  text-align: initial;
  overflow: hidden !important;
}
```

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[p1535]'s content area.

Once the animation has ended, the user agent is expected[p1488] to increment the marquee current loop index[p1537]. If the element is still turned on[p1536] after this, then the user agent is expected[p1488] to restart the animation.

If the element's behavior[p1536] attribute is in the alternate[p1536] state

When the marquee current loop index[p1537] is even (or zero), slide the contents of the element in the direction described by the direction[p1536] attribute as defined below, such that it begins flush with the start side of the marquee[p1535], and ends flush with the end side of the marquee[p1535].

When the marquee current loop index[p1537] is odd, slide the contents of the element in the opposite direction than that described by the direction[p1536] attribute as defined below, such that it begins flush with the end side of the marquee[p1535], and ends flush with the start side of the marquee[p1535].

Example

For example, if the direction[p1536] attribute is left[p1536] (the default), then the contents would with their right edge flush with the right inner edge of the marquee[p1535]'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[p1535]'s content area.

Once the animation has ended, the user agent is expected[p1488] to increment the marquee current loop index[p1537]. If the element is still turned on[p1536] after this, then the user agent is expected[p1488] to continue the animation.

The direction[p1536] attribute has the meanings described in the following table:

direction[p1536] attribute state	Direction of animation	Start edge	End edge	Opposite direction
left[p1536]	← Right to left	Right	Left	→ Left to Right
right[p1536]	→ Left to Right	Left	Right	← Right to left
up[p1536]	↑ Up (Bottom to Top)	Bottom	Top	↓ Down (Top to Bottom)
down[p1536]	↓ Down (Top to Bottom)	Top	Bottom	↑ Up (Bottom to Top)

In any case, the animation should proceed such that there is a delay given by the marquee scroll interval[p1536] between each frame, and such that the content moves at most the distance given by the marquee scroll distance[p1536] with each frame.

When a marquee[p1535] element has a `bgcolor` attribute set, the value is expected[p1488] to be parsed using the rules for parsing a legacy color value[p98], and if that does not return failure, the user agent is expected[p1488] to treat the attribute as a presentational hint[p1489] setting the element's 'background-color' property to the resulting color.

The `width` and `height` attributes on a marquee[p1535] element map to the dimension properties[p1489] 'width' and 'height' on the element respectively.

The natural height of a marquee[p1535] element with its direction[p1536] attribute in the up[p1536] or down[p1536] states is 200 CSS pixels.

The `vspace` attribute of a marquee[p1535] element maps to the dimension properties[p1489] 'margin-top' and 'margin-bottom' on the element. The `hspace` attribute of a marquee[p1535] element maps to the dimension properties[p1489] 'margin-left' and 'margin-right' on the element.

15.5.14 The meter[p616] element §[p15] 16

```css
@namespace "http://www.w3.org/1999/xhtml";

meter { appearance: auto; }
```

The meter[p616] element is a devolvable widget. Its native appearance is expected[p1488] to render as an 'inline-block' box with a 'block-size' of '1em' and an 'inline-size' of '5em', a 'vertical-align' of '-0.2em', and with its contents depicting a gauge.

When this element has a horizontal writing mode[p1511], the depiction is expected[p1488] to be of a horizontal gauge. Its minimum value is on the right if the 'direction' property has a computed value of 'rtl', and on the left otherwise. When this element has a vertical writing mode[p1511], it is expected[p1488] to depict a vertical gauge. Its minimum value is on the bottom if the 'direction' property has a computed

value of 'rtl', and on the top otherwise.

User agents are expected[p1488] 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[p616] element.

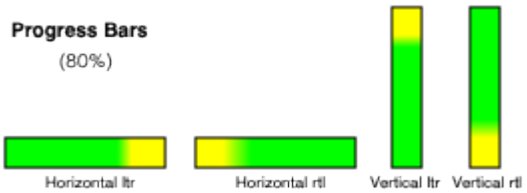
Need to detail the primitive appearance.

15.5.15 The progress[p614] element §[p15 17]

```css
@namespace "http://www.w3.org/1999/xhtml";

progress { appearance: auto; }
```

The progress[p614] element is a devolvable widget. Its native appearance is expected[p1488] to render as an 'inline-block' box with a 'block-size' of '1em' and an 'inline-size' of '10em', and a 'vertical-align' of '-0.2em'.



When this element has a horizontal writing mode[p1511], the element is expected[p1488] to be depicted as a horizontal progress bar. The start is on the right and the end is 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 this element has a vertical writing mode[p1511], it is expected[p1488] to be depicted as a vertical progress bar. The start is on the bottom and the end is on the top if the 'direction' property on this element has a computed value of 'rtl', and with the start on the top and the end on the bottom otherwise.

User agents are expected[p1488] to use a presentation consistent with platform conventions for progress bars. In particular, user agents are expected[p1488] to use different presentations for determinate and indeterminate progress bars. User agents are also expected[p1488] to vary the presentation based on the dimensions of the element.

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[p614] element.

Need to detail the primitive appearance.

15.5.16 The select[p590] element §[p15 17]

The select[p590] element is an element with default preferred size, and user agents are expected[p1488] to apply the 'field-sizing' CSS property to select[p590] elements.

A select[p590] element is either a **list box** or a **drop-down box**, depending on its attributes.

A select[p590] element whose multiple[p591] attribute is present is expected[p1488] to render as a multi-select list box[p1517] if its display size[p593] is greater than 1. If the select[p590] element has the multiple[p591] attribute and a display size[p593] of 1, then it may render as a multi-select drop-down box[p1517] if the platform supports it; otherwise as a multi-select list box[p1517].

A select[p590] element whose multiple[p591] attribute is absent is expected[p1488] to render as a single-select drop-down box[p1517] if its display size[p593] is 1, or as a single-select list box[p1517] if its display size[p593] is greater than 1.

When the element renders as a list box[p1517], it is a devolvable widget expected[p1488] to render as an 'inline-block' box. The inline size of its intrinsic size is the width of the `select`'s labels[p1519] plus the width of a scrollbar. The block size of its intrinsic size is determined by the following steps:

1. If the 'field-sizing' property on the element has a computed value of 'content', return the height necessary to contain all rows for items.
2. If the `size`[p591] attribute is absent or it has no valid value, return the height necessary to contain four rows.
3. Otherwise, return the height necessary to contain as many rows for items as given by the element's display size[p593].

A `select`[p590] element which is being rendered as a drop-down box[p1517] is expected[p1488] to render as an 'inline-block' box. The inline size of its intrinsic size is the width of the `select`'s labels[p1519]. If the 'field-sizing' property on the element has a computed value of 'content', the inline size of the intrinsic size depends on the shown text. The shown text is typically the label of an `option`[p601] of which selectedness[p603] is set to true.

When the element renders as a drop-down box[p1517], it is a devolvable widget. Its appearance in the devolved state, as well as its appearance when the computed value of the element's 'appearance' property is `'menulist-button'`, is that of a drop-down box, including a "drop-down button", but not necessarily rendered using a native control of the host operating system. In such a state, CSS properties such as 'color', 'background-color', and 'border' should not be disregarded (as is generally permissible when rendering an element according to its native appearance).

In either case (list box[p1517] or drop-down box[p1517]), the element's items are expected[p1488] to be the element's list of options[p593], with the element's `optgroup`[p600] element children providing headers for groups of options where applicable.

`select`[p590] elements which render as a drop-down box[p1517] support a base appearance in addition to native appearance and primitive appearance.

`select`[p590] elements which render as a drop-down box[p1517] without the `multiple`[p591] attribute or as a list box[p1517] support a base appearance in addition to native appearance and primitive appearance.

The `select`[p590] element's select popover[p1518] supports a base appearance and a native appearance. The select popover[p1518] can only be rendered with base appearance if its associated `select`[p590] is being rendered with base appearance.

When a `select`[p590] is being rendered as a drop-down box[p1517] with base appearance, it is expected[p1488] to render with a shadow tree that contains the following elements:

1. A **select button slot**, which is a `slot`[p712] element. It is appended to the `select`[p590]'s shadow root as the first child. It is expected[p1488] to take the first child element of the `select`[p590] if the first child element is a `button`[p585].
2. A **select fallback button text**, which is a `div`[p266] element. It is appended to the select button slot[p1518].
3. A **select popover**, which is a `div`[p266] element. It is appended to the `select`[p590]'s shadow root as the second child, after the select button slot[p1518]. The `select`[p590] element's '::picker' pseudo-element is the select popover[p1518] if the provided argument is `select`.
4. A **select popover slot**, which is a `slot`[p712] element. It is appended to the select popover[p1518]. It is expected[p1488] to take all child nodes of the `select`[p590] except for the first child `button`[p585], which is taken by the select button slot[p1518].

Note

Since base appearance is determined by computing style, it isn't possible to swap this DOM structure when switching appearance. Implementations can always include the DOM structure for base appearance when the `select`[p590] is rendered as a drop-down box[p1517] and then choose to include or exclude it from the layout tree in order to control whether it gets rendered or not.

Note

The select popover[p1518] is only rendered when it is opted in to base appearance separately from the `select`[p590] element. Otherwise, a native picker is used.

When a `select`[p590] is being rendered as a list box[p1517] with base appearance, it is expected to render with a shadow tree that contains a **select list box slot**, which is a `slot`[p712] element. The select list box slot[p1518] is appended to the `select`[p590]'s shadow root as the first child. The select list box slot[p1518] is expected to take all children of the `select`[p590] element.

The select popover[p1518]'s implicit anchor element is its associated `select`[p590] element.

When a select element is being rendered with native appearance or primitive appearance, or the select element is being rendered as a list box, the '::picker' pseudo-element and the '::picker-icon' pseudo-element do not apply.

The '::picker' pseudo-element is not rendered when it has native appearance or primitive appearance.

The '::checkmark' pseudo-element only applies to option elements which are being rendered with base appearance.

An optgroup element has an internal shadow tree with at least one slot element.

An optgroup element is **rendered with base appearance** if it has a nearest ancestor select whose contents are being rendered with base appearance.

If an optgroup element is not being rendered with base appearance, then it is expected to be rendered by displaying the element's label and children.

If an optgroup element is being rendered with base appearance, then it is expected to render with a shadow tree that contains the following elements:

1. An **optgroup legend slot**, which is a slot element. It is appended to the optgroup's shadow root as the first child. It is expected to take the first child element of the optgroup if the first child element is a legend element.
2. An **optgroup label element**, which is a div element. It is appended to the optgroup legend slot. It is expected to contain a Text node whose data is the result of getting an optgroup element's label given the optgroup element.
3. An **optgroup contents slot**, which is a slot element. It is appended to the optgroup's shadow root after the optgroup legend slot. It is expected to take all remaining child nodes of the optgroup element.

To determine if a **select's contents are being rendered with base appearance**, given a select element *select*:

1. If *select* is being rendered as a list box with base appearance, then return true.
2. If *select* is being rendered as a drop-down box with base appearance, and its select popover is being rendered with base appearance, and *select* does not have the multiple attribute set, then return true.
3. Return false.

An option element has an internal shadow tree with a slot element.

An option element is **rendered with base appearance** if it has a nearest ancestor select and the select's contents are being rendered with base appearance.

If an option does not have a nearest ancestor select, then it is expected to render its children.

Otherwise, if an option element is not being rendered with base appearance, then it is expected to be rendered by displaying the result of getting an option element's label given the option and true.

Otherwise, it is expected to render with a shadow tree that contains the following elements:

1. An **option contents slot**, which is a slot element. It is appended to the option's shadow root as the first child. If the option element has a label attribute whose value is not the empty string, then the option contents slot is expected to not take any nodes. Otherwise, it is expected to take all child nodes of the option element.

   Note

   When the option contents slot does not take any elements, the option label element is rendered as fallback content.

2. An **option label element**, which is a span element. It is appended to the option contents slot. It is expected to contain a Text node whose data is the value of the option's label attribute.

Each sequence of one or more child hr element siblings may be rendered as a single separator.

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 the select has the multiple attribute and is being rendered as a drop-down box, then the width should also be wide enough to accomodate the text rendered in the select with any combination of options selected.

Note

The width of the select's labels[p1519] has an accomodation for multiple[p591] because some implementations use special text to represent multiple options being selected, such as "2 selected." In this case, the select needs to be wide enough to render "2 selected" in addition to the individual option[p601]s.

If a select[p590] element contains a placeholder label option[p591], the user agent is expected[p1488] to render that option[p601] 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[p591] from being explicitly selected by the user. When the placeholder label option[p591]'s selectedness[p603] is true, the control is expected[p1488] to be displayed in a fashion that indicates that no valid option is currently selected.

User agents are expected[p1488] to render the labels in a select[p590] 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.

Need to detail the native appearance and primitive appearance.

CSS
```
@namespace "http://www.w3.org/1999/xhtml";

option {
  display: block;
}

optgroup {
  display: block;
  font-weight: bolder;
}
```

The following styles are expected[p1488] to apply to select[p590] elements when they are being rendered as a drop-down box[p1517]:

CSS
```
@namespace "http://www.w3.org/1999/xhtml";

select {
  align-content: center;
}
```

The following styles are expected[p1488] to apply to select[p590] elements when they are being rendered with native appearance or primitive appearance:

CSS
```
@namespace "http://www.w3.org/1999/xhtml";

select {
  letter-spacing: initial;
  word-spacing: initial;
  line-height: initial;
}
```

The following styles are expected[p1488] to apply to select[p590] elements when they are being rendered as a drop-down box[p1517] with native appearance or primitive appearance:

CSS
```
@namespace "http://www.w3.org/1999/xhtml";

select {
  display: inline-block;
}
```

The following styles are expected[p1488] to apply to select[p590] elements when they are being rendered with base appearance:

CSS
```
@namespace "http://www.w3.org/1999/xhtml";

select {
```

```
  background-color: transparent;
  border: 1px solid currentColor;
  user-select: none;
  box-sizing: border-box;
}

select option:enabled:hover {
  background-color: color-mix(currentColor 10%, transparent);
}
select option:enabled:active {
  background-color: color-mix(currentColor 20%, transparent);
}
select option:disabled {
  color: color-mix(currentColor 50%, transparent);
}

select option {
  min-inline-size: 24px;
  min-block-size: max(24px, 1lh);
  padding-inline: 0.5em;
  padding-block-end: 0;
  display: flex;
  align-items: center;
  gap: 0.5em;
  white-space: nowrap;
}

select option::checkmark {
  content: '\2713' / '';
}
select option:not(:checked)::checkmark {
  visibility: hidden;
}

select optgroup {
  display: block;
  font-weight: bolder;
}

select optgroup option {
  font-weight: normal;
}

select optgroup legend {
  padding-inline: 0.5em;
  min-block-size: 1lh;
}
```

The following styles are expected[p1488] to apply to select[p590] elements when they are being rendered as a drop-down box[p1517] with base appearance:

CSS
```
@namespace "http://www.w3.org/1999/xhtml";

select {
  border-radius: 0.5em;
  padding-block: 0.25em;
  padding-inline: 0.5em;
  min-block-size: calc-size(auto, max(size, 24px, 1lh));
  min-inline-size: calc-size(auto, max(size, 24px));
  display: inline-flex;
  gap: 0.5em;
  border-radius: 0.5em;
```

```
  field-sizing: content !important;
}

select > button:first-child {
  all: unset;
  display: contents;
}

select:enabled:hover {
  background-color: color-mix(currentColor 10%, transparent);
}
select:enabled:active {
  background-color: color-mix(currentColor 20%, transparent);
}
select:disabled {
  color: color-mix(currentColor 50%, transparent);
}

::picker(select) {
  box-sizing: border-box;
  border: 1px solid;
  padding: 0;
  color: CanvasText;
  background-color: Canvas;
  margin: 0;
  inset: auto;
  min-inline-size: anchor-size(self-inline);
  max-block-size: stretch;
  overflow: auto;
  position-area: self-block-end span-self-inline-end;
  position-try-order: most-block-size;
  position-try-fallbacks:
    self-block-start span-self-inline-end,
    self-block-end span-self-inline-start,
    self-block-start span-self-inline-start;
}

select::picker-icon {
  content: counter(fake-counter-name, disclosure-open);
  display: block;
  margin-inline-start: auto;
}
```

The following styles are expected[p1488] to apply to select[p590] elements when they are being rendered as a list box[p1517] with base appearance:

CSS
```
@namespace "http://www.w3.org/1999/xhtml";

select {
  overflow: auto;
  display: inline-block;
  block-size: calc(max(24px, 1lh) * attr(size type(<integer>), 4));
}
```

The following styles are expected[p1488] to apply to the optgroup label element[p1519]:

- padding-inline: 0.5em
- min-block-size: 1lh

15.5.17 The textarea element

The textarea element is a devolvable widget 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.

The textarea element is an element with default preferred size, and user agents are expected to apply the 'field-sizing' CSS property to textarea elements.

If the 'field-sizing' property on the element has a computed value of 'content', the intrinsic size is determined from the text which the element shows. The text is either a raw value or a short hint specified by the placeholder attribute. User agents may take the text caret size into account in the intrinsic size. Otherwise, its intrinsic size is computed from textarea effective width and textarea effective height (as defined below).

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.)

The **textarea effective height** of a textarea element is the height in CSS pixels of the number of lines given by 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'.

Need to detail the native appearance and primitive appearance.

15.6 Frames and framesets

User agents are expected to render frameset elements as a box with the height and width of the viewport, with a surface rendered according to the following layout algorithm:

1. The *cols* and *rows* variables are lists of zero or more pairs consisting of a number and a unit, the unit being one of *percentage*, *relative*, and *absolute*.

   Use the rules for parsing a list of dimensions to parse the value of the element's cols attribute, if there is one. Let *cols* be the result, or an empty list if there is no such attribute.

   Use the rules for parsing a list of dimensions to parse the value of the element's rows attribute, if there is one. Let *rows* be the result, or an empty list if there is no such attribute.

2. For any of the entries in *cols* or *rows* that have the number zero and the unit *relative*, change the entry's number to one.

3. If *cols* has no entries, then add a single entry consisting of the value 1 and the unit *relative* to *cols*.

   If *rows* has no entries, then add a single entry consisting of the value 1 and the unit *relative* to *rows*.

4. Invoke the algorithm defined below to convert a list of dimensions to a list of pixel values using *cols* as the input list, and the width of the surface that the frameset is being rendered into, in CSS pixels, as the input dimension. Let *sized cols* be the resulting list.

   Invoke the algorithm defined below to convert a list of dimensions to a list of pixel values using *rows* as the input list, and the height of the surface that the frameset is being rendered into, in CSS pixels, as the input dimension. Let *sized rows* be the resulting list.

5. Split the surface into a grid of *w*×*h* rectangles, where *w* is the number of entries in *sized cols* and *h* is the number of entries in *sized rows*.

   Size the columns so that each column in the grid is as many CSS pixels wide as the corresponding entry in the *sized cols* list.

   Size the rows so that each row in the grid is as many CSS pixels high as the corresponding entry in the *sized rows* list.

6. Let *children* be the list of `frame`[p1537] and `frameset`[p1537] elements that are children of the `frameset`[p1537] element for which the algorithm was invoked.

7. For each row of the grid of rectangles created in the previous step, from top to bottom, run these substeps:

    1. For each rectangle in the row, from left to right, run these substeps:

        1. If there are any elements left in *children*, take the first element in the list, and assign it to the rectangle.

            If this is a `frameset`[p1537] element, then recurse the entire `frameset`[p1537] layout algorithm for that `frameset`[p1537] element, with the rectangle as the surface.

            Otherwise, it is a `frame`[p1537] element; render its content navigable[p1038], positioned and sized to fit the rectangle.

        2. If there are any elements left in *children*, remove the first element from *children*.

8. If the `frameset`[p1537] element has a border[p1524], draw an outer set of borders around the rectangles, using the element's frame border color[p1524].

    For each rectangle, if there is an element assigned to that rectangle, and that element has a border[p1524], draw an inner set of borders around that rectangle, using the element's frame border color[p1524].

    For each (visible) border that does not abut a rectangle that is assigned a `frame`[p1537] element with a `noresize` attribute (including rectangles in further nested `frameset`[p1537] elements), the user agent is expected[p1488] to allow the user to move the border, resizing the rectangles within, keeping the proportions of any nested `frameset`[p1537] grids.

    A `frameset`[p1537] or `frame`[p1537] 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`[p1537] element, then return true if *that* element has a border[p1524], and false if it does not.

    4. Otherwise, return true.

    The **frame border color** of a `frameset`[p1537] or `frame`[p1537] 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[p98] to that attribute's value does not return failure, then return the color so obtained.

    2. Otherwise, if the element has a parent element that is a `frameset`[p1537] element, then return the frame border color[p1524] 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 *total percentage* be the sum of all the numbers in *input list* whose unit is *percentage*.

    Let *total relative* be the sum of all the numbers in *input list* whose unit is *relative*.

    Let *total absolute* be the sum of all the numbers in *input list* whose unit is *absolute*.

    Let *remaining space* be the value of *input dimension*.

4. If *total absolute* is greater than *remaining space*, then for each entry in *input list* whose unit is *absolute*, set the corresponding value in *output list* to the number of the entry in *input list* multiplied by *remaining space* and divided by *total absolute*. Then, set *remaining space* to zero.

Otherwise, for each entry in *input list* whose unit is *absolute*, set the corresponding value in *output list* to the number of the entry in *input list*. Then, decrement *remaining space* by *total absolute*.

5. If *total percentage* multiplied by the *input dimension* and divided by 100 is greater than *remaining space*, then for each entry in *input list* whose unit is *percentage*, set the corresponding value in *output list* to the number of the entry in *input list* multiplied by *remaining space* and divided by *total percentage*. Then, set *remaining space* to zero.

   Otherwise, for each entry in *input list* whose unit is *percentage*, set the corresponding value in *output list* to the number of the entry in *input list* multiplied by the *input dimension* and divided by 100. Then, decrement *remaining space* by *total percentage* multiplied by the *input dimension* and divided by 100.

6. For each entry in *input list* whose unit is *relative*, set the corresponding value in *output list* to the number of the entry in *input list* multiplied by *remaining space* and divided by *total relative*.

7. Return *output list*.

User agents working with integer values for frame widths (as opposed to user agents that can lay frames out with subpixel accuracy) are expected to distribute the remainder first to the last entry whose unit is *relative*, then equally (not proportionally) to each entry whose unit is *percentage*, then equally (not proportionally) to each entry whose unit is *absolute*, and finally, failing all else, to the last entry.

The contents of a `frame` element that does not have a `frameset` parent are expected to be rendered as transparent black; the user agent is expected to not render its content navigable in this case, and its content navigable 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 navigable 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 navigables to the URLs indicated by the `cite` attributes on `q`, `blockquote`, `ins`, and `del` elements.

User agents may surface hyperlinks created by `link` elements in their user interface, 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:

```
<p dir="rtl" lang="he">
 <label>
  :בחר שפת תכנות
>  select>
   <option dir="ltr">C++</option>
   <option dir="ltr">C#</option>
   <option dir="ltr">FreePascal</option>
   <option dir="ltr">F#</option>
  </select>
 </label>
</p>
```

If the `select` element was rendered as a drop down box, a correct rendering would ensure that the punctuation was the same both in the drop down, and in the box showing the current selection.

Example

The directionality of attributes depends on the attribute and on the element's `dir` attribute, as the following example

demonstrates. Consider this markup:

```
<table>
 <tr>
  <th abbr="(א" dir=ltr>A
  <th abbr="(א" dir=rtl>A
  <th abbr="(א" dir=auto>A
</table>
```

If the abbr[p514] attributes are rendered, e.g. in a tooltip or other user interface, the first will have a left parenthesis (because the direction is 'ltr'), the second will have a right parenthesis (because the direction is 'rtl'), and the third will have a right parenthesis (because the direction is determined *from the attribute value* to be 'rtl').

However, if instead the attribute was not a directionality-capable attribute[p170], the results would be different:

```
<table>
 <tr>
  <th data-abbr="(א" dir=ltr>A
  <th data-abbr="(א" dir=rtl>A
  <th data-abbr="(א" dir=auto>A
</table>
```

In this case, if the user agent were to expose the `data-abbr` attribute in the user interface (e.g. in a debugging environment), the last case would be rendered with a *left* parenthesis, because the direction would be determined from the element's contents.

A string provided by a script (e.g. the argument to window.alert()[p1259]) is expected[p1488] to be treated as an independent set of one or more bidirectional algorithm paragraphs when displayed, as defined by the bidirectional algorithm, including, for instance, supporting the paragraph-breaking behavior of U+000A LINE FEED (LF) characters. For the purposes of determining the paragraph level of such text in the bidirectional algorithm, this specification does *not* provide a higher-level override of rules P2 and P3. [BIDI][p1579]

When necessary, authors can enforce a particular direction for a given paragraph by starting it with the Unicode U+200E LEFT-TO-RIGHT MARK or U+200F RIGHT-TO-LEFT MARK characters.

Example

Thus, the following script:

```
alert('\u05DC\u05DE\u05D3 HTML \u05D4\u05D9\u05D5\u05DD!')
```

...would always result in a message reading "!היום HTML למד" (not "למד HTML היום!"), regardless of the language of the user agent interface or the direction of the page or any of its elements.

Example

For a more complex example, consider the following script:

```
/* Warning: this script does not handle right-to-left scripts correctly */
var s;
if (s = prompt('What is your name?')) {
  alert(s + '! Ok, Fred, ' + s + ', and Wilma will get the car.');
}
```

When the user enters "Kitty", the user agent would alert "Kitty! Ok, Fred, Kitty, and Wilma will get the car.". However, if the user enters "ع أ فهم", 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: ".and Wilma will get the car ,فهم أ ع ,Ok, Fred !فهم أ ع"

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 §p15 28

User agents are expected[p1488] to allow the user to request the opportunity to **obtain a physical form** (or a representation of a physical form) of a Document[p135]. For example, selecting the option to print a page or convert it to PDF format. [PDF][p1585]

When the user actually obtains a physical form[p1528] (or a representation of a physical form) of a Document[p135], the user agent is expected[p1488] to create a new rendering of the Document[p135] for the print media.

15.9 Unstyled XML documents §p15 28

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[p135] is an unstyled document[p1528], the user agent is expected[p1488] to render an unstyled document view[p1528].

A Document[p135] is an **unstyled document** while it matches the following conditions:

- The Document[p135] has no author style sheets (whether referenced by HTTP headers, processing instructions, elements like link[p184], inline elements like style[p207], or any other mechanism).
- None of the elements in the Document[p135] have any presentational hints[p1489].
- None of the elements in the Document[p135] have any style attributes.
- None of the elements in the Document[p135] are in any of the following namespaces: HTML namespace, SVG namespace, MathML namespace.
- The Document[p135] has no focusable area[p874] (e.g. from XLink) other than the viewport.
- The Document[p135] has no hyperlinks[p313] (e.g. from XLink).
- There exists no script[p1152] whose settings object[p1152]'s global object[p1145] is a Window[p965] object with this Document[p135] as its associated Document[p966].
- None of the elements in the Document[p135] 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[p135] 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[p135] stops being an unstyled document[p1528], 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 §[p15 29]

16.1 Obsolete but conforming features §[p15 29]

Features listed in this section will trigger warnings in conformance checkers.

Authors should not specify a `border`[p1534] attribute on an `img`[p359] element. If the attribute is present, its value must be the string "`0`". CSS should be used instead.

Authors should not specify a `charset`[p1531] attribute on a `script`[p685] 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`[p1533] attribute on a `script`[p685] element. If the attribute is present, its value must be an ASCII case-insensitive match for the string "`JavaScript`" and either the `type`[p686] 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`[p686] attribute.

Authors should not specify a value for the `type`[p686] attribute on `script`[p685] 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`[p1533] attribute on a `style`[p207] element. If the attribute is present, its value must be an ASCII case-insensitive match for "`text/css`[p1577]".

Authors should not specify the `name`[p1531] attribute on `a`[p267] elements. If the attribute is present, its value must not be the empty string and must neither be equal to the value of any of the IDs in the element's tree other than the element's own ID, if any, nor be equal to the value of any of the other `name`[p1531] attributes on `a`[p267] 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`[p162] attribute should be used instead.

Authors should not, but may despite requirements to the contrary elsewhere in this specification, specify the `maxlength`[p569] and `size`[p570] attributes on `input`[p539] elements whose `type`[p541] attributes are in the Number[p556] state. One valid reason for using these attributes regardless is to help legacy user agents that do not support `input`[p539] elements with `type="number"` to still render the text control with a useful width.

16.1.1 Warnings for obsolete but conforming features §[p15 29]

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`[p1534] attribute on an `img`[p359] element if its value is the string "`0`".
- The presence of a `charset`[p1531] attribute on a `script`[p685] element if its value is an ASCII case-insensitive match for "`utf-8`".
- The presence of a `language`[p1533] attribute on a `script`[p685] element if its value is an ASCII case-insensitive match for the string "`JavaScript`" and if there is no `type`[p686] attribute or there is and its value is an ASCII case-insensitive match for the string "`text/javascript`".
- The presence of a `type`[p1533] attribute on a `script`[p685] element if its value is a JavaScript MIME type essence match.
- The presence of a `type`[p1533] attribute on a `style`[p207] element if its value is an ASCII case-insensitive match for "`text/css`[p1577]".
- The presence of a `name`[p1531] attribute on an `a`[p267] element, if its value is not the empty string.
- The presence of a `maxlength`[p569] attribute on an `input`[p539] element whose `type`[p541] attribute is in the Number[p556] state.

- The presence of a size[p570] attribute on an input[p539] element whose type[p541] attribute is in the Number[p556] 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 §[p1530]

Elements in the following list are entirely obsolete, and must not be used by authors:

applet

Use embed[p413] or object[p416] instead.

acronym

Use abbr[p279] instead.

bgsound

Use audio[p425] instead.

dir

Use ul[p249] instead.

frame[p1537]

frameset[p1537]

noframes

Either use iframe[p404] and CSS instead, or use server-side includes to generate complete pages with the various invariant parts merged in.

isindex

Use an explicit form[p532] and text control[p546] 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][p1588]

listing

Use pre[p242] and code[p297] instead.

menuitem

To implement a custom context menu, use script to handle the contextmenu event.

nextid

Use GUIDs instead.

noembed

Use object[p416] instead of embed[p413] when fallback is necessary.

param

Use the data[p417] attribute of the object[p416] element to set the URL of the external resource.

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 "<" and "&" respectively.

basefont

big

blink

center

font

marquee

multicol

nobr

spacer

tt

Use appropriate elements or CSS instead.

Where the tt element would have been used for marking up keyboard input, consider the kbd element; for variables, consider the var element; for computer code, consider the code element; and for computer output, consider the samp element.

Similarly, if the big element is being used to denote a heading, consider using the h1 element; if it is being used for marking up important passages, consider the 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

`usemap` on `object` elements

Use the `img` element 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.

declare on object elements

Repeat the object element completely each time the resource is to be reused.

standby on object elements

Optimize the linked resource so that it loads quickly or, at least, incrementally.

typemustmatch on object elements

Avoid using object elements with untrusted resources.

language on script elements (except as noted in the previous section)

Omit the attribute for JavaScript; for data blocks, use the type attribute instead.

event on script elements

for on script elements

Use DOM events mechanisms to register event listeners. [DOM]

type on style elements (except as noted in the previous section)

Omit the attribute for CSS; for data blocks, use script as the container instead of style.

datapagesize on table elements

Unnecessary. Omit it altogether.

summary on table elements

Use one of the techniques for describing tables given in the table section instead.

abbr on td elements

Use text that begins in an unambiguous and terse manner, and include any more elaborate text after that. The title 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 (which has an abbr attribute).

axis on td and th elements

Use the scope attribute on the relevant th.

scope on td elements

Use th elements for heading cells.

datasrc on a, button, div, frame, iframe, img, input, label, legend, marquee, object, option, select, span, table, and textarea elements

datafld on a, button, div, fieldset, frame, iframe, img, input, label, legend, marquee, object, select, span, and textarea elements

dataformatas on button, div, input, label, legend, marquee, object, option, select, span, and table elements

Use script and a mechanism such as XMLHttpRequest to populate the page dynamically. [XHR]

dropzone on all elements

Use script to handle the dragenter and dragover events instead.

alink on **body**[p212] elements
bgcolor on **body**[p212] elements
bottommargin on **body**[p212] elements
leftmargin on **body**[p212] elements
link on **body**[p212] elements
marginheight on **body**[p212] elements
marginwidth on **body**[p212] elements
rightmargin on **body**[p212] elements
text on **body**[p212] elements
topmargin on **body**[p212] elements
vlink on **body**[p212] elements
clear on **br**[p310] elements
align on **caption**[p504] elements
align on **col**[p506] elements
char on **col**[p506] elements
charoff on **col**[p506] elements
valign on **col**[p506] elements
width on **col**[p506] elements
align on **div**[p266] elements
compact on **dl**[p253] elements
align on **embed**[p413] elements
hspace on **embed**[p413] elements
vspace on **embed**[p413] elements
align on **hr**[p241] elements
color on **hr**[p241] elements
noshade on **hr**[p241] elements
size on **hr**[p241] elements
width on **hr**[p241] elements
align on **h1**[p224]—**h6**[p224] elements
align on **iframe**[p404] elements
allowtransparency on **iframe**[p404] elements
frameborder on **iframe**[p404] elements
framespacing on **iframe**[p404] elements
hspace on **iframe**[p404] elements
marginheight on **iframe**[p404] elements
marginwidth on **iframe**[p404] elements
scrolling on **iframe**[p404] elements
vspace on **iframe**[p404] elements
align on **input**[p539] elements
border on **input**[p539] elements
hspace on **input**[p539] elements
vspace on **input**[p539] elements
align on **img**[p359] elements
border on **img**[p359] elements (except as noted in the previous section)
hspace on **img**[p359] elements
vspace on **img**[p359] elements
align on **legend**[p623] elements
type on **li**[p251] elements
compact on **menu**[p250] elements
align on **object**[p416] elements
border on **object**[p416] elements
hspace on **object**[p416] elements
vspace on **object**[p416] elements

compact on **ol**[p247] elements
align on **p**[p238] elements
width on **pre**[p242] elements
align on **table**[p496] elements
bgcolor on **table**[p496] elements
border on **table**[p496] elements
bordercolor on **table**[p496] elements
cellpadding on **table**[p496] elements
cellspacing on **table**[p496] elements
frame on **table**[p496] elements
height on **table**[p496] elements
rules on **table**[p496] elements
width on **table**[p496] elements
align on **tbody**[p507], **thead**[p508], and **tfoot**[p509] elements
char on **tbody**[p507], **thead**[p508], and **tfoot**[p509] elements
charoff on **tbody**[p507], **thead**[p508], and **tfoot**[p509] elements
height on **thead**[p508], **tbody**[p507], and **tfoot**[p509] elements
valign on **tbody**[p507], **thead**[p508], and **tfoot**[p509] elements
align on **td**[p512] and **th**[p513] elements
bgcolor on **td**[p512] and **th**[p513] elements
char on **td**[p512] and **th**[p513] elements
charoff on **td**[p512] and **th**[p513] elements
height on **td**[p512] and **th**[p513] elements
nowrap on **td**[p512] and **th**[p513] elements
valign on **td**[p512] and **th**[p513] elements
width on **td**[p512] and **th**[p513] elements
align on **tr**[p510] elements
bgcolor on **tr**[p510] elements
char on **tr**[p510] elements
charoff on **tr**[p510] elements
height on **tr**[p510] elements
valign on **tr**[p510] elements
compact on **ul**[p249] elements
type on **ul**[p249] elements
background on **body**[p212], **table**[p496], **thead**[p508], **tbody**[p507], **tfoot**[p509], **tr**[p510], **td**[p512], and **th**[p513] elements

Use CSS instead.

16.3 Requirements for implementations §[p15 35]

16.3.1 The marquee element §[p15 35]

The marquee[p1535] element is a presentational element that animates content. CSS transitions and animations are a more appropriate mechanism. [CSSANIMATIONS][p1580] [CSSTRANSITIONS][p1581]

The marquee[p1535] element must implement the HTMLMarqueeElement[p1535] interface.

IDL

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

  [CEReactions, Reflect] attribute DOMString behavior;
  [CEReactions, Reflect] attribute DOMString bgColor;
  [CEReactions, Reflect] attribute DOMString direction;
```

```
  [CEReactions, Reflect] attribute DOMString height;
  [CEReactions, Reflect] attribute unsigned long hspace;
  [CEReactions] attribute long loop;
  [CEReactions, Reflect, ReflectDefault=6] attribute unsigned long scrollAmount;
  [CEReactions, Reflect, ReflectDefault=85] attribute unsigned long scrollDelay;
  [CEReactions, Reflect] attribute boolean trueSpeed;
  [CEReactions, Reflect] attribute unsigned long vspace;
  [CEReactions, Reflect] attribute DOMString width;

  undefined start();
  undefined stop();
};
```

A marquee[p1535] element can be **turned on** or **turned off**. When it is created, it is turned on[p1536].

When the **start()** method is called, the marquee[p1535] element must be turned on[p1536].

When the **stop()** method is called, the marquee[p1535] element must be turned off[p1536].

The **behavior** content attribute on marquee[p1535] elements is an enumerated attribute[p80] with the following keywords and states (all non-conforming):

Keyword	State
scroll	**scroll**
slide	**slide**
alternate	**alternate**

The attribute's *missing value default*[p80] and *invalid value default*[p80] are both the scroll[p1536] state.

The **direction** content attribute on marquee[p1535] elements is an enumerated attribute[p80] with the following keywords and states (all non-conforming):

Keyword	State
left	**left**
right	**right**
up	**up**
down	**down**

The attribute's *missing value default*[p80] and *invalid value default*[p80] are both the left[p1536] state.

The **truespeed** content attribute on marquee[p1535] elements is a boolean attribute[p79].

A marquee[p1535] 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[p81] 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[p1536] attribute, and the *delay* value is less than 60, then let *delay* be 60 instead.
3. The marquee scroll interval[p1536] is *delay*, interpreted in milliseconds.

A marquee[p1535] 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[p81] 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 greater than or equal to the element's marquee loop count, turn off the marquee element.

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, Reflect] attribute DOMString cols;
  [CEReactions, Reflect] attribute DOMString rows;
};
HTMLFrameSetElement includes WindowEventHandlers;
```

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 content navigable similar to the iframe element, but rendered within a frameset element.

The frame HTML element post-connection steps, given *insertedNode*, are:

1. If *insertedNode*'s node document's browsing context is null, then return.
2. Create a new child navigable for *insertedNode*.
3. Process the frame attributes for *insertedNode*, with *initialInsertion* set to true.

The frame HTML element removing steps, given *removedNode*, are to destroy a child navigable given *removedNode*.

Whenever a frame element with a non-null content navigable has its `src` attribute set, changed, or removed, the user agent must process the frame attributes.

To **process the frame attributes** for an element *element*, with an optional boolean ***initialInsertion*** (default false):

1. Let *url* be the result of running the shared attribute processing steps for iframe and frame elements given *element* and *initialInsertion*.
2. If *url* is null, then return.
3. If *url* matches about:blank and *initialInsertion* is true:

1. Fire an event named load[p1575] at *element*.

2. Return.

4. Navigate an iframe or frame[p408] given *element*, *url*, the empty string, null, and *initialInsertion*.

The frame[p1537] element potentially delays the load event[p409].

The frame[p1537] element must implement the HTMLFrameElement[p1538] interface.

```
IDL
[Exposed=Window]
interface HTMLFrameElement : HTMLElement {
  [HTMLConstructor] constructor();

  [CEReactions, Reflect] attribute DOMString name;
  [CEReactions, Reflect] attribute DOMString scrolling;
  [CEReactions, ReflectURL] attribute USVString src;
  [CEReactions, Reflect] attribute DOMString frameBorder;
  [CEReactions, ReflectURL] attribute USVString longDesc;
  [CEReactions, Reflect] attribute boolean noResize;
  readonly attribute Document? contentDocument;
  readonly attribute WindowProxy? contentWindow;

  [CEReactions, Reflect] attribute [LegacyNullToEmptyString] DOMString marginHeight;
  [CEReactions, Reflect] attribute [LegacyNullToEmptyString] DOMString marginWidth;
};
```

The **contentDocument** getter steps are to return this's content document[p1039].

The **contentWindow** getter steps are to return this's content window[p1039].

16.3.3 Other elements, attributes and APIs §[p1538]

User agents must treat acronym[p1530] elements in a manner equivalent to abbr[p279] elements in terms of semantics and for purposes of rendering.

```
IDL
partial interface HTMLAnchorElement {
  [CEReactions, Reflect] attribute DOMString coords;
  [CEReactions, Reflect] attribute DOMString charset;
  [CEReactions, Reflect] attribute DOMString name;
  [CEReactions, Reflect] attribute DOMString rev;
  [CEReactions, Reflect] attribute DOMString shape;
};
```

```
IDL
partial interface HTMLAreaElement {
  [CEReactions, Reflect] attribute boolean noHref;
};
```

```
IDL
partial interface HTMLBodyElement {
  [CEReactions, Reflect] attribute [LegacyNullToEmptyString] DOMString text;
  [CEReactions, Reflect] attribute [LegacyNullToEmptyString] DOMString link;
  [CEReactions, Reflect] attribute [LegacyNullToEmptyString] DOMString vLink;
  [CEReactions, Reflect] attribute [LegacyNullToEmptyString] DOMString aLink;
  [CEReactions, Reflect] attribute [LegacyNullToEmptyString] DOMString bgColor;
  [CEReactions, Reflect] attribute DOMString background;
```

```
};
```

Note

The background[p1538] IDL attribute does not use ReflectURL[p114] or USVString for historical reasons.

IDL

```
partial interface HTMLBRElement {
  [CEReactions, Reflect] attribute DOMString clear;
};
```

IDL

```
partial interface HTMLTableCaptionElement {
  [CEReactions, Reflect] attribute DOMString align;
};
```

IDL

```
partial interface HTMLTableColElement {
  [CEReactions, Reflect] attribute DOMString align;
  [CEReactions, Reflect="char"] attribute DOMString ch;
  [CEReactions, Reflect="charoff"] attribute DOMString chOff;
  [CEReactions, Reflect] attribute DOMString vAlign;
  [CEReactions, Reflect] attribute DOMString width;
};
```

User agents must treat dir[p1530] elements in a manner equivalent to ul[p249] elements in terms of semantics and for purposes of rendering.

The dir[p1530] element must implement the HTMLDirectoryElement[p1539] interface.

IDL

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

  [CEReactions, Reflect] attribute boolean compact;
};
```

IDL

```
partial interface HTMLDivElement {
  [CEReactions, Reflect] attribute DOMString align;
};
```

IDL

```
partial interface HTMLDListElement {
  [CEReactions, Reflect] attribute boolean compact;
};
```

IDL

```
partial interface HTMLEmbedElement {
  [CEReactions, Reflect] attribute DOMString align;
  [CEReactions, Reflect] attribute DOMString name;
};
```

The font[p1531] element must implement the HTMLFontElement[p1540] interface.

IDL

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

  [CEReactions, Reflect] attribute [LegacyNullToEmptyString] DOMString color;
  [CEReactions, Reflect] attribute DOMString face;
  [CEReactions, Reflect] attribute DOMString size;
};
```

IDL

```
partial interface HTMLHeadingElement {
  [CEReactions, Reflect] attribute DOMString align;
};
```

Note

*The **profile** IDL attribute on head[p180] elements (with the HTMLHeadElement[p180] interface) is intentionally omitted. Unless so required by another applicable specification[p77], implementations would therefore not support this attribute. (It is mentioned here as it was defined in a previous version of* DOM*.)*

IDL

```
partial interface HTMLHRElement {
  [CEReactions, Reflect] attribute DOMString align;
  [CEReactions, Reflect] attribute DOMString color;
  [CEReactions, Reflect] attribute boolean noShade;
  [CEReactions, Reflect] attribute DOMString size;
  [CEReactions, Reflect] attribute DOMString width;
};
```

IDL

```
partial interface HTMLHtmlElement {
  [CEReactions, Reflect] attribute DOMString version;
};
```

IDL

```
partial interface HTMLIFrameElement {
  [CEReactions, Reflect] attribute DOMString align;
  [CEReactions, Reflect] attribute DOMString scrolling;
  [CEReactions, Reflect] attribute DOMString frameBorder;
  [CEReactions, ReflectURL] attribute USVString longDesc;

  [CEReactions, Reflect] attribute [LegacyNullToEmptyString] DOMString marginHeight;
  [CEReactions, Reflect] attribute [LegacyNullToEmptyString] DOMString marginWidth;
};
```

IDL

```
partial interface HTMLImageElement {
  [CEReactions, Reflect] attribute DOMString name;
  [CEReactions, ReflectURL] attribute USVString lowsrc;
  [CEReactions, Reflect] attribute DOMString align;
  [CEReactions, Reflect] attribute unsigned long hspace;
  [CEReactions, Reflect] attribute unsigned long vspace;
  [CEReactions, ReflectURL] attribute USVString longDesc;

  [CEReactions, Reflect] attribute [LegacyNullToEmptyString] DOMString border;
};
```

IDL

```
partial interface HTMLInputElement {
  [CEReactions, Reflect] attribute DOMString align;
  [CEReactions, Reflect] attribute DOMString useMap;
};
```

IDL

```
partial interface HTMLLegendElement {
  [CEReactions, Reflect] attribute DOMString align;
};
```

IDL

```
partial interface HTMLLIElement {
  [CEReactions, Reflect] attribute DOMString type;
};
```

IDL

```
partial interface HTMLLinkElement {
  [CEReactions, Reflect] attribute DOMString charset;
  [CEReactions, Reflect] attribute DOMString rev;
  [CEReactions, Reflect] attribute DOMString target;
};
```

User agents must treat `listing`[p1530] elements in a manner equivalent to `pre`[p242] elements in terms of semantics and for purposes of rendering.

IDL

```
partial interface HTMLMenuElement {
  [CEReactions, Reflect] attribute boolean compact;
};
```

IDL

```
partial interface HTMLMetaElement {
  [CEReactions, Reflect] attribute DOMString scheme;
};
```

User agents may treat the `scheme`[p1532] content attribute on the `meta`[p196] element as an extension of the element's `name`[p197] content attribute when processing a `meta`[p196] element with a `name`[p197] attribute whose value is one that the user agent recognizes as supporting the `scheme`[p1532] attribute.

User agents are encouraged to ignore the `scheme`[p1532] attribute and instead process the value given to the metadata name as if it had been specified for each expected value of the `scheme`[p1532] attribute.

Example

For example, if the user agent acts on `meta`[p196] elements with `name`[p197] attributes having the value "eGMS.subject.keyword", and knows that the `scheme`[p1532] attribute is used with this metadata name, then it could take the `scheme`[p1532] attribute into account, acting as if it was an extension of the `name`[p197] attribute. Thus the following two `meta`[p196] elements could be treated as two elements giving values for two different metadata names, one consisting of a combination of "eGMS.subject.keyword" and "LGCL", and the other consisting of a combination of "eGMS.subject.keyword" and "ORLY":

```
<!-- this markup is invalid -->
<meta name="eGMS.subject.keyword" scheme="LGCL" content="Abandoned vehicles">
<meta name="eGMS.subject.keyword" scheme="ORLY" content="Mah car: kthxbye">
```

The suggested processing of this markup, however, would be equivalent to the following:

```
<meta name="eGMS.subject.keyword" content="Abandoned vehicles">
<meta name="eGMS.subject.keyword" content="Mah car: kthxbye">
```

IDL

```
partial interface HTMLObjectElement {
  [CEReactions, Reflect] attribute DOMString align;
  [CEReactions, Reflect] attribute DOMString archive;
  [CEReactions, Reflect] attribute DOMString code;
  [CEReactions, Reflect] attribute boolean declare;
  [CEReactions, Reflect] attribute unsigned long hspace;
  [CEReactions, Reflect] attribute DOMString standby;
  [CEReactions, Reflect] attribute unsigned long vspace;
  [CEReactions, ReflectURL] attribute DOMString codeBase;
  [CEReactions, Reflect] attribute DOMString codeType;
  [CEReactions, Reflect] attribute DOMString useMap;

  [CEReactions, Reflect] attribute [LegacyNullToEmptyString] DOMString border;
};
```

IDL

```
partial interface HTMLOListElement {
  [CEReactions, Reflect] attribute boolean compact;
};
```

IDL

```
partial interface HTMLParagraphElement {
  [CEReactions, Reflect] attribute DOMString align;
};
```

The param[p1530] element must implement the HTMLParamElement[p1542] interface.

IDL

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

  [CEReactions, Reflect] attribute DOMString name;
  [CEReactions, Reflect] attribute DOMString value;
  [CEReactions, Reflect] attribute DOMString type;
  [CEReactions, Reflect] attribute DOMString valueType;
};
```

User agents must treat plaintext[p1530] elements in a manner equivalent to pre[p242] elements in terms of semantics and for purposes of rendering. (The parser has special behavior for this element, though.)

IDL

```
partial interface HTMLPreElement {
  [CEReactions, Reflect] attribute long width;
};
```

IDL

```
partial interface HTMLStyleElement {
  [CEReactions, Reflect] attribute DOMString type;
};
```

IDL

```
partial interface HTMLScriptElement {
  [CEReactions, Reflect] attribute DOMString charset;
  [CEReactions, Reflect] attribute DOMString event;
  [CEReactions, Reflect="for"] attribute DOMString htmlFor;
};
```

IDL

```
partial interface HTMLTableElement {
  [CEReactions, Reflect] attribute DOMString align;
  [CEReactions, Reflect] attribute DOMString border;
  [CEReactions, Reflect] attribute DOMString frame;
  [CEReactions, Reflect] attribute DOMString rules;
  [CEReactions, Reflect] attribute DOMString summary;
  [CEReactions, Reflect] attribute DOMString width;

  [CEReactions, Reflect] attribute [LegacyNullToEmptyString] DOMString bgColor;
  [CEReactions, Reflect] attribute [LegacyNullToEmptyString] DOMString cellPadding;
  [CEReactions, Reflect] attribute [LegacyNullToEmptyString] DOMString cellSpacing;
};
```

IDL

```
partial interface HTMLTableSectionElement {
  [CEReactions, Reflect] attribute DOMString align;
  [CEReactions, Reflect="char"] attribute DOMString ch;
  [CEReactions, Reflect="charoff"] attribute DOMString chOff;
  [CEReactions, Reflect] attribute DOMString vAlign;
};
```

IDL

```
partial interface HTMLTableCellElement {
  [CEReactions, Reflect] attribute DOMString align;
  [CEReactions, Reflect] attribute DOMString axis;
  [CEReactions, Reflect] attribute DOMString height;
  [CEReactions, Reflect] attribute DOMString width;

  [CEReactions, Reflect="char"] attribute DOMString ch;
  [CEReactions, Reflect="charoff"] attribute DOMString chOff;
  [CEReactions, Reflect] attribute boolean noWrap;
  [CEReactions, Reflect] attribute DOMString vAlign;

  [CEReactions, Reflect] attribute [LegacyNullToEmptyString] DOMString bgColor;
};
```

IDL

```
partial interface HTMLTableRowElement {
  [CEReactions, Reflect] attribute DOMString align;
  [CEReactions, Reflect="char"] attribute DOMString ch;
  [CEReactions, Reflect="charoff"] attribute DOMString chOff;
  [CEReactions, Reflect] attribute DOMString vAlign;

  [CEReactions, Reflect] attribute [LegacyNullToEmptyString] DOMString bgColor;
};
```

IDL

```
partial interface HTMLUListElement {
  [CEReactions, Reflect] attribute boolean compact;
  [CEReactions, Reflect] attribute DOMString type;
```

```
};
```

User agents must treat xmp[p1531] elements in a manner equivalent to pre[p242] 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[p135] object listed in the first column of the following table must reflect[p108] the content attribute on the body element[p144] with the name given in the corresponding cell in the second column on the same row, if the body element[p144] is a body[p212] element (as opposed to a frameset[p1537] element). When there is no body element[p144] or if it is a frameset[p1537] element, the attributes must instead return the empty string on getting and do nothing on setting.

IDL attribute	Content attribute
fgColor	text[p1534]
linkColor	link[p1534]
vlinkColor	vlink[p1534]
alinkColor	alink[p1534]
bgColor	bgcolor[p1534]

The **anchors** attribute must return an HTMLCollection rooted at the Document[p135] node, whose filter matches only a[p267] elements with name[p1531] attributes.

The **applets** attribute must return an HTMLCollection rooted at the Document[p135] 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[p116] rooted at the Document[p135] node, whose filter matches all elements.

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[p965] interface must return an instance of the External[p1545] interface:

IDL

```
[Exposed=Window]
interface External {
  undefined AddSearchProvider();
  undefined IsSearchProviderInstalled();
};
```

The **AddSearchProvider()** and **IsSearchProviderInstalled()** methods must do nothing.

17 IANA considerations §[p15 46]

17.1 text/html §[p15 46]

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[p206] 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][p1582]

Encoding considerations:

8bit (see the section on character encoding declarations[p206])

Security considerations:

Entire novels have been written about the security considerations that apply to HTML documents. Many are listed in this document, to which the reader is referred for more details. Some general concerns bear mentioning here, however:

HTML is scripted language, and has a large number of APIs (some of which are described in this document). Script can expose the user to potential risks of information leakage, credential leakage, cross-site scripting attacks, cross-site request forgeries, and a host of other problems. While the designs in this specification are intended to be safe if implemented correctly, a full implementation is a massive undertaking and, as with any software, user agents are likely to have security bugs.

Even without scripting, there are specific features in HTML which, for historical reasons, are required for broad compatibility with legacy content but that expose the user to unfortunate security problems. In particular, the `img`[p359] 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[p939] 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[p939] 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`[p1546] type asserts that the resource is an HTML document using the HTML syntax[p1356].

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][p1584]

File extension(s):

"`html`" and "`htm`" are commonly, but certainly not exclusively, used as the extension for HTML documents.

Macintosh file type code(s):
TEXT

Person & email address to contact for further information:
Ian Hickson <ian@hixie.ch>

Intended usage:
Common

Restrictions on usage:
No restrictions apply.

Author:
Ian Hickson <ian@hixie.ch>

Change controller:
W3C

Fragments used with `text/html`[p1546] resources either refer to the indicated part[p1104] of the corresponding `Document`[p135], or provide state information for in-page scripts.

17.2 `multipart/x-mixed-replace` §[p1547]

This registration is for community review and will be submitted to the IESG for review, approval, and registration with IANA.

Type name:
multipart

Subtype name:
x-mixed-replace

Required parameters:

- `boundary` (defined in RFC2046) [RFC2046][p1585]

Optional parameters:
No optional parameters.

Encoding considerations:
binary

Security considerations:
Subresources of a `multipart/x-mixed-replace`[p1547] resource can be of any type, including types with non-trivial security implications such as `text/html`[p1546].

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`[p1577]. [RFC2046][p1585]

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`[p1547] 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`[p1547] resources apply to each body part as defined by the type used by that body part.

17.3 `application/xhtml+xml` §[p1548]

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`[p1577] [RFC7303][p1586]

Optional parameters:

Same as for `application/xml`[p1577] [RFC7303][p1586]

Encoding considerations:

Same as for `application/xml`[p1577] [RFC7303][p1586]

Security considerations:

Same as for `application/xml`[p1577] [RFC7303][p1586]

Interoperability considerations:

Same as for `application/xml`[p1577] [RFC7303][p1586]

Published specification:

Labeling a resource with the `application/xhtml+xml`[p1548] 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][p1588] [XMLNS][p1588]

Applications that use this media type:

Same as for `application/xml`[p1577] [RFC7303][p1586]

Additional information:

Magic number(s):

Same as for `application/xml`[p1577] [RFC7303][p1586]

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`[p1548] resources have the same semantics as with any XML MIME type. [RFC7303][p1586]

17.4 `text/ping` §[p15 49]

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[p325], 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`[p1549] 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`[p314] attribute.

Author:

Ian Hickson <ian@hixie.ch>

Change controller:

W3C

Fragments have no meaning with `text/ping`[p1549] resources.

17.5 `application/microdata+json` §[p1550]

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`[p1577] [JSON][p1583]

Optional parameters:

Same as for `application/json`[p1577] [JSON][p1583]

Encoding considerations:

8bit (always UTF-8)

Security considerations:

Same as for `application/json`[p1577] [JSON][p1583]

Interoperability considerations:

Same as for `application/json`[p1577] [JSON][p1583]

Published specification:

Labeling a resource with the `application/microdata+json`[p1550] 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. [JSON][p1583]

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`[p1577] [JSON][p1583]

File extension(s):

Same as for `application/json`[p1577] [JSON][p1583]

Macintosh file type code(s):

Same as for `application/json`[p1577] [JSON][p1583]

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/microdata+json[p1550] resources have the same semantics as when used with application/json[p1577] (namely, at the time of writing, no semantics at all). [JSON][p1583]

17.6 application/speculationrules+json §[p1551]

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[p1577] [JSON][p1583]

Optional parameters:

Same as for application/json[p1577] [JSON][p1583]

Encoding considerations:

8bit (always UTF-8)

Security considerations:

Same as for application/json[p1577] [JSON][p1583]

Interoperability considerations:

Same as for application/json[p1577] [JSON][p1583]

Published specification:

Labeling a resource with the application/microdata+json[p1550] type asserts that the resource is a JSON text that follows the speculation rule set authoring requirements[p1119]. Thus, the relevant specifications are *JSON* and this specification. [JSON][p1583]

Applications that use this media type:

Web browsers.

Additional information:

Magic number(s):

Same as for application/json[p1577] [JSON][p1583]

File extension(s):

Same as for application/json[p1577] [JSON][p1583]

Macintosh file type code(s):

Same as for application/json[p1577] [JSON][p1583]

Person & email address to contact for further information:

Domenic Denicola <d@domenic.me>

Intended usage:

Common

Restrictions on usage:

No restrictions apply.

Author:

Domenic Denicola <d@domenic.me>

Change controller:

WHATWG

Fragments used with application/speculationrules+json[p1551] resources have the same semantics as when used with application/json[p1577] (namely, at the time of writing, no semantics at all). [JSON][p1583]

17.7 text/event-stream §[p15 52]

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][p1582]

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>

Change controller:

W3C

Fragments have no meaning with `text/event-stream`[p1552] resources.

17.8 web+ scheme prefix §[p15 53]

This section describes a convention for use with the IANA URI scheme registry. It does not itself register a specific scheme. [RFC7595][p1586]

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[p1265]

Index §p15 54

The following sections only cover conforming elements and features.

Elements §p15 54

This section is non-normative.

List of elements

Element	Description	Categories	Parents†	Children	Attributes	Interface
a p267	Hyperlink	flow p157; phrasing p157*; interactive p158; palpable p158	phrasing p157	transparent p159*	globals p162; href p314; target p314; download p314; ping p314; rel p314; hreflang p316; type p316; referrerpolicy p314	HTMLAnchorElement p268
abbr p279	Abbreviation	flow p157; phrasing p157; palpable p158	phrasing p157	phrasing p157	globals p162	HTMLElement p149
address p230	Contact information for a page or article p214 element	flow p157; palpable p158	flow p157	flow p157*	globals p162	HTMLElement p149
area p489	Hyperlink or dead area on an image map	flow p157; phrasing p157	phrasing p157*	empty	globals p162; alt p490; coords p490; shape p490; href p314; target p314; download p314; ping p314; rel p314; referrerpolicy p314	HTMLAreaElement p489
article p214	Self-contained syndicatable or reusable composition	flow p157; sectioning p157; palpable p158	flow p157	flow p157	globals p162	HTMLElement p149
aside p222	Sidebar for tangentially related content	flow p157; sectioning p157; palpable p158	flow p157	flow p157	globals p162	HTMLElement p149
audio p425	Audio player	flow p157; phrasing p157; embedded p158; interactive p158; palpable p158*	phrasing p157	source p355*; track p427*; transparent p159*	globals p162; src p433; crossorigin p433; preload p446; autoplay p452; loading p431; loop p450; muted p483; controls p481	HTMLAudioElement p426
b p303	Keywords	flow p157; phrasing p157; palpable p158	phrasing p157	phrasing p157	globals p162	HTMLElement p149
base p182	Base URL and default target navigable p1035 for hyperlinks p314 and forms p632	metadata p156	head p180	empty	globals p162; href p183; target p183	HTMLBaseElement p182
bdi p307	Text directionality isolation	flow p157; phrasing p157; palpable p158	phrasing p157	phrasing p157	globals p162	HTMLElement p149
bdo p309	Text directionality formatting	flow p157; phrasing p157; palpable p158	phrasing p157	phrasing p157	globals p162	HTMLElement p149
blockquote p244	A section quoted from another source	flow p157; palpable p158	flow p157	flow p157	globals p162; cite p245	HTMLQuoteElement p244
body p212	Document body	none	html p179	flow p157	globals p162; onafterprint p1215; onbeforeprint p1215; onbeforeunload p1215;	HTMLBodyElement p213

Element	Description	Categories	Parents†	Children	Attributes	Interface
					onhashchange[p1215]; onlanguagechange[p1215]; onmessage[p1215]; onmessageerror[p1215]; onoffline[p1215]; ononline[p1215]; onpageswap[p1215]; onpagehide[p1215]; onpagereveal[p1215]; onpageshow[p1215]; onpopstate[p1215]; onrejectionhandled[p1215]; onstorage[p1215]; onunhandledrejection[p1215]; onunload[p1215]	
br[p310]	Line break, e.g. in poem or postal address	flow[p157]; phrasing[p157]	phrasing[p157]	empty	globals[p162]	HTMLBRElement[p310]
button[p585]	Button control	flow[p157]; phrasing[p157]; interactive[p158]; listed[p532]; labelable[p532]; submittable[p532]; form-associated[p532]; palpable[p158]	phrasing[p157]; select[p590]*	phrasing[p157]*	globals[p162]; command[p586]; commandfor[p586]; disabled[p630]; form[p627]; formaction[p631]; formenctype[p632]; formmethod[p631]; formnovalidate[p632]; formtarget[p632]; name[p628]; popovertarget[p935]; popovertargetaction[p935]; type[p586]; value[p588]	HTMLButtonElement[p585]
canvas[p713]	Scriptable bitmap canvas	flow[p157]; phrasing[p157]; embedded[p158]; palpable[p158]	phrasing[p157]	transparent[p159]	globals[p162]; width[p715]; height[p715]	HTMLCanvasElement[p714]
caption[p504]	Table caption	none	table[p496]	flow[p157]*	globals[p162]	HTMLTableCaptionElement[p504]
cite[p275]	Title of a work	flow[p157]; phrasing[p157]; palpable[p158]	phrasing[p157]	phrasing[p157]	globals[p162]	HTMLElement[p149]
code[p297]	Computer code	flow[p157]; phrasing[p157]; palpable[p158]	phrasing[p157]	phrasing[p157]	globals[p162]	HTMLElement[p149]
col[p506]	Table column	none	colgroup[p505]	empty	globals[p162]; span[p507]	HTMLTableColElement[p506]
colgroup[p505]	Group of columns in a table	none	table[p496]	col[p506]*; template[p705]*	globals[p162]; span[p506]	HTMLTableColElement[p506]
data[p289]	Machine-readable equivalent	flow[p157]; phrasing[p157]; palpable[p158]	phrasing[p157]	phrasing[p157]	globals[p162]; value[p289]	HTMLDataElement[p289]
datalist[p598]	Container for options for combo box control[p576]	flow[p157]; phrasing[p157]	phrasing[p157]	phrasing[p157]*; option[p601]*; script-supporting elements[p159]*	globals[p162]	HTMLDataListElement[p599]
dd[p258]	Content for corresponding dt[p257] element(s)	none	dl[p253]; div[p266]*	flow[p157]	globals[p162]	HTMLElement[p149]
del[p351]	A removal from the document	flow[p157]; phrasing[p157]*; palpable[p158]	phrasing[p157]	transparent[p159]	globals[p162]; cite[p352]; datetime[p352]	HTMLModElement[p352]
details[p666]	Disclosure control for hiding details	flow[p157]; interactive[p158]; palpable[p158]	flow[p157]	summary[p672]*; flow[p157]	globals[p162]; name[p667]; open[p667]	HTMLDetailsElement[p666]
dfn[p278]	Defining instance	flow[p157]; phrasing[p157]; palpable[p158]	phrasing[p157]	phrasing[p157]*	globals[p162]	HTMLElement[p149]
dialog[p675]	Dialog box or window	flow[p157]	flow[p157]	flow[p157]	globals[p162]; open[p677]	HTMLDialogElement[p676]
div[p266]	Generic flow container, or container for name-value groups in dl[p253]	flow[p157]; palpable[p158]	flow[p157]; dl[p253]; option[p601]; optgroup[p600]; select[p590]	flow[p157]*	globals[p162]	HTMLDivElement[p266]

Element	Description	Categories	Parents†	Children	Attributes	Interface
	elements					
dl	Association list consisting of zero or more name-value groups	flow; palpable	flow	dt*; dd*; div*; script-supporting elements	globals	HTMLDListElement
dt	Legend for corresponding dd element(s)	none	dl; div*	flow*	globals	HTMLElement
em	Stress emphasis	flow; phrasing; palpable	phrasing	phrasing	globals	HTMLElement
embed	Plugin	flow; phrasing; embedded; interactive; palpable	phrasing	empty	globals; src; type; width; height; any*	HTMLEmbedElement
fieldset	Group of form controls	flow; listed; form-associated; palpable	flow	legend*; flow	globals; disabled; form; name	HTMLFieldSetElement
figcaption	Caption for figure	none	figure	flow	globals	HTMLElement
figure	Figure with optional caption	flow; palpable	flow	figcaption*; flow	globals	HTMLElement
footer	Footer for a page or section	flow; palpable	flow	flow*	globals	HTMLElement
form	User-submittable form	flow; palpable	flow	flow*	globals; accept-charset; action; autocomplete; enctype; method; name; novalidate; rel; target	HTMLFormElement
h1, h2, h3, h4, h5, h6	Heading	flow; heading; palpable	legend; summary; flow	phrasing	globals	HTMLHeadingElement
head	Container for document metadata	none	html	metadata content*	globals	HTMLHeadElement
header	Introductory or navigational aids for a page or section	flow; palpable	flow	flow*	globals	HTMLElement
hgroup	Heading container	flow; palpable	legend; summary; flow	h1; h2; h3; h4; h5; h6; p; script-supporting elements	globals	HTMLElement
hr	Thematic break	flow	flow; select	empty	globals	HTMLHRElement
html	Root element	none	none*	head*; body*	globals	HTMLHtmlElement
i	Alternate voice	flow; phrasing; palpable	phrasing	phrasing	globals	HTMLElement
iframe	Child navigable	flow; phrasing; embedded; interactive; palpable	phrasing	empty	globals; src; srcdoc; name; sandbox; allow; allowfullscreen; width; height; referrerpolicy; loading	HTMLIFrameElement
img	Image	flow; phrasing; embedded;	phrasing; picture	empty	globals; alt; src; srcset; sizes; crossorigin; usemap; ismap; controls; width;	HTMLImageElement

Element	Description	Categories	Parents†	Children	Attributes	Interface
		interactive[p158]*; form-associated[p532]; palpable[p158]			height[p495]; referrerpolicy[p361]; decoding[p361]; loading[p361]; fetchpriority[p361]	
input[p539]	Form control	flow[p157]; phrasing[p157]; interactive[p158]*; listed[p532]; labelable[p532]; submittable[p532]; resettable[p532]; form-associated[p532]; palpable[p158]*	phrasing[p157]	empty	globals[p162]; accept[p563]; alpha[p559]; alt[p567]; autocomplete[p633]; checked[p543]; colorspace[p560]; dirname[p629]; disabled[p630]; form[p627]; formaction[p631]; formenctype[p632]; formmethod[p631]; formnovalidate[p632]; formtarget[p632]; height[p495]; list[p576]; max[p574]; maxlength[p569]; min[p574]; minlength[p569]; multiple[p572]; name[p628]; pattern[p573]; placeholder[p578]; popovertarget[p935]; popovertargetaction[p935]; readonly[p570]; required[p571]; size[p570]; src[p566]; step[p575]; type[p541]; value[p543]; width[p495]	HTMLInputElement[p540]
ins[p350]	An addition to the document	flow[p157]; phrasing[p157]*; palpable[p158]	phrasing[p157]	transparent[p159]	globals[p162]; cite[p352]; datetime[p352]	HTMLModElement[p352]
kbd[p300]	User input	flow[p157]; phrasing[p157]; palpable[p158]	phrasing[p157]	phrasing[p157]	globals[p162]	HTMLElement[p149]
label[p536]	Caption for a form control	flow[p157]; phrasing[p157]; interactive[p158]; palpable[p158]	phrasing[p157]	phrasing[p157]*	globals[p162]; for[p537]	HTMLLabelElement[p537]
legend[p623]	Caption for fieldset[p620]	none	fieldset[p620]; optgroup[p600]	phrasing[p157]*; heading content[p157]	globals[p162]	HTMLLegendElement[p624]
li[p251]	List item	none	ol[p247]; ul[p249]; menu[p250]*	flow[p157]	globals[p162]; value[p251]*	HTMLLIElement[p251]
link[p184]	Link metadata	metadata[p156]; flow[p157]*; phrasing[p157]*	head[p180]; noscript[p703]*; phrasing[p157]*	empty	globals[p162]; href[p185]; crossorigin[p186]; rel[p185]; as[p188]; media[p186]; hreflang[p186]; type[p186]; sizes[p188]; imagesrcset[p187]; imagesizes[p187]; referrerpolicy[p187]; integrity[p186]; blocking[p188]; color[p188]; disabled[p188]; fetchpriority[p189]	HTMLLinkElement[p185]
main[p262]	Container for the dominant contents of the document	flow[p157]; palpable[p158]	flow[p157]*	flow[p157]	globals[p162]	HTMLElement[p149]
map[p487]	Image map[p491]	flow[p157]; phrasing[p157]*; palpable[p158]	phrasing[p157]	transparent[p159]; area[p489]*	globals[p162]; name[p488]	HTMLMapElement[p488]
mark[p305]	Highlight	flow[p157]; phrasing[p157]; palpable[p158]	phrasing[p157]	phrasing[p157]	globals[p162]	HTMLElement[p149]
MathML math	MathML root	flow[p157]; phrasing[p157]; embedded[p158]; palpable[p158]	phrasing[p157]	per [MATHML][p1584]	per [MATHML][p1584]	Element
menu[p250]	Menu of commands	flow[p157]; palpable[p158]*	flow[p157]	li[p251]; script-supporting elements[p159]	globals[p162]	HTMLMenuElement[p250]
meta[p196]	Text metadata	metadata[p156]; flow[p157]*; phrasing[p157]*	head[p180]; noscript[p703]*; phrasing[p157]*	empty	globals[p162]; name[p197]; http-equiv[p202]; content[p197]; charset[p197]; media[p197]	HTMLMetaElement[p197]
meter[p616]	Gauge	flow[p157]; phrasing[p157]; labelable[p532]; palpable[p158]	phrasing[p157]	phrasing[p157]*	globals[p162]; value[p617]; min[p617]; max[p617]; low[p617]; high[p617]; optimum[p617]	HTMLMeterElement[p616]
nav[p219]	Section with navigational links	flow[p157]; sectioning[p157]; palpable[p158]	flow[p157]	flow[p157]	globals[p162]	HTMLElement[p149]
noscript[p703]	Fallback	metadata[p156];	head[p180]*;	varies*	globals[p162]	HTMLElement[p149]

Element	Description	Categories	Parents†	Children	Attributes	Interface
	content for script	flow p157; phrasing p157	phrasing p157*; select p590; optgroup p600			
object p416	Image, child navigable p1039, or plugin p48	flow p157; phrasing p157; embedded p158; interactive p158*; listed p532; form-associated p532; palpable p158	phrasing p157	transparent p159	globals p162; data p417; type p417; name p417; form p627; width p495; height p495	HTMLObjectElement p416
ol p247	Ordered list	flow p157; palpable p158*	flow p157	li p251; script-supporting elements p159	globals p162; reversed p248; start p248; type p248	HTMLOListElement p247
optgroup p600	Group of options in a list box	none	select p590; div p266*	option p601*; script-supporting elements p159*; noscript p703*; div p266*; legend p623*	globals p162; disabled p600; label p601	HTMLOptGroupElement p600
option p601	Option in a list box or combo box control	none	select p590; datalist p598; optgroup p600; div p266*	text p158*; div p266*; phrasing p157*	globals p162; disabled p602; label p603; selected p603; value p603	HTMLOptionElement p602
output p611	Calculated output value	flow p157; phrasing p157; listed p532; labelable p532; resettable p532; form-associated p532; palpable p158	phrasing p157	phrasing p157	globals p162; for p612; form p627; name p628	HTMLOutputElement p612
p p238	Paragraph	flow p157; palpable p158	flow p157	phrasing p157	globals p162	HTMLParagraphElement p238
picture p355	Image	flow p157; phrasing p157; embedded p158; palpable p158	phrasing p157	source p355*; one img p359; script-supporting elements p159	globals p162	HTMLPictureElement p355
pre p242	Block of preformatted text	flow p157; palpable p158	flow p157	phrasing p157	globals p162	HTMLPreElement p243
progress p614	Progress bar	flow p157; phrasing p157; labelable p532; palpable p158	phrasing p157	phrasing p157*	globals p162; value p614; max p614	HTMLProgressElement p614
q p277	Quotation	flow p157; phrasing p157; palpable p158	phrasing p157	phrasing p157	globals p162; cite p277	HTMLQuoteElement p244
rp p288	Parenthesis for ruby annotation text	none	ruby p281	text p158	globals p162	HTMLElement p149
rt p287	Ruby annotation text	none	ruby p281	phrasing p157	globals p162	HTMLElement p149
ruby p281	Ruby annotation(s)	flow p157; phrasing p157; palpable p158	phrasing p157	phrasing p157; rt p287; rp p288*	globals p162	HTMLElement p149
s p274	Inaccurate text	flow p157; phrasing p157; palpable p158	phrasing p157	phrasing p157	globals p162	HTMLElement p149
samp p299	Computer output	flow p157; phrasing p157; palpable p158	phrasing p157	phrasing p157	globals p162	HTMLElement p149
script p685	Embedded script	metadata p156; flow p157; phrasing p157;	head p180; phrasing p157; script-	script, data, or script documentation*	globals p162; src p686; type p686; nomodule p687; async p687; defer p687; crossorigin p688; integrity p688;	HTMLScriptElement p686

Element	Description	Categories	Parents†	Children	Attributes	Interface
		script-supporting p159	supporting p159		referrerpolicy p688; blocking p688; fetchpriority p688	
search p264	Container for search controls	flow p157; palpable p158	flow p157	flow p157	globals p162	HTMLElement p149
section p216	Generic document or application section	flow p157; sectioning p157; palpable p158	flow p157	flow p157	globals p162	HTMLElement p149
select p590	List box control	flow p157; phrasing p157; interactive p158; listed p532; labelable p532; submittable p532; resettable p532; form-associated p532; palpable p158	phrasing p157	option p601*; optgroup p600*; hr p241*; script-supporting elements p159*; noscript p703*; div p266*; button p585*	globals p162; autocomplete p633; disabled p630; form p627; multiple p591; name p628; required p591; size p591	HTMLSelectElement p590
selectedcontent p624	Mirrors content from an option p601	none	button p585	empty	globals p162	HTMLSelectedContentElement p624
slot p712	Shadow tree slot	flow p157; phrasing p157	phrasing p157	transparent p159	globals p162; name p712	HTMLSlotElement p712
small p272	Side comment	flow p157; phrasing p157; palpable p158	phrasing p157	phrasing p157	globals p162	HTMLElement p149
source p355	Image source for img p359 or media source for video p420 or audio p425	none	picture p355; video p420; audio p425	empty	globals p162; type p356; media p356; src p357; srcset p356; sizes p357; width p495; height p495	HTMLSourceElement p356
span p309	Generic phrasing container	flow p157; phrasing p157; palpable p158	phrasing p157	phrasing p157	globals p162	HTMLSpanElement p310
strong p271	Importance	flow p157; phrasing p157; palpable p158	phrasing p157	phrasing p157	globals p162	HTMLElement p149
style p207	Embedded styling information	metadata p156	head p180; noscript p703*	text*	globals p162; media p208; blocking p209	HTMLStyleElement p208
sub p301	Subscript	flow p157; phrasing p157; palpable p158	phrasing p157	phrasing p157	globals p162	HTMLElement p149
summary p672	Caption for details p666	none	details p666	phrasing p157; heading content p157	globals p162	HTMLElement p149
sup p301	Superscript	flow p157; phrasing p157; palpable p158	phrasing p157	phrasing p157	globals p162	HTMLElement p149
SVG svg	SVG root	flow p157; phrasing p157; embedded p158; palpable p158	phrasing p157	per [SVG] p1586	per [SVG] p1586	SVGSVGElement
table p496	Table	flow p157; palpable p158	flow p157	caption p504*; colgroup p505*; thead p508*; tbody p507*; tfoot p509*; tr p510*; script-supporting elements p159	globals p162	HTMLTableElement p496
tbody p507	Group of rows in a table	none	table p496	tr p510; script-supporting elements p159	globals p162	HTMLTableSectionElement p507
td p512	Table cell	none	tr p510	flow p157	globals p162; colspan p515; rowspan p515; headers p515	HTMLTableCellElement p512

Element	Description	Categories	Parents†	Children	Attributes	Interface
template p705	Template	metadata p156; flow p157; phrasing p157; script-supporting p159	metadata p156; phrasing p157; script-supporting p159; colgroup p505*	empty	globals p162; for p706; shadowrootmode p706; shadowrootdelegatesfocus p706; shadowrootserializable p706; shadowrootslotassignment p706; shadowrootclonable p706; shadowrootcustomelementregistry p706	HTMLTemplateElement p705
textarea p606	Multiline text controls	flow p157; phrasing p157; interactive p158; listed p532; labelable p532; submittable p532; resettable p532; form-associated p532; palpable p158	phrasing p157	text p158	globals p162; autocomplete p633; cols p609; dirname p629; disabled p630; form p627; maxlength p609; minlength p609; name p628; placeholder p610; readonly p608; required p610; rows p609; wrap p609	HTMLTextAreaElement p607
tfoot p509	Group of footer rows in a table	none	table p496	tr p510; script-supporting elements p159	globals p162	HTMLTableSectionElement p507
th p513	Table header cell	interactive p158*	tr p510	flow p157*	globals p162; colspan p515; rowspan p515; headers p515; scope p514; abbr p514	HTMLTableCellElement p512
thead p508	Group of heading rows in a table	none	table p496	tr p510; script-supporting elements p159	globals p162	HTMLTableSectionElement p507
time p290	Machine-readable equivalent of date- or time-related data	flow p157; phrasing p157; palpable p158	phrasing p157	phrasing p157	globals p162; datetime p290	HTMLTimeElement p290
title p181	Document title	metadata p156	head p180	text p158*	globals p162	HTMLTitleElement p181
tr p510	Table row	none	table p496; thead p508; tbody p507; tfoot p509	th p513*; td p512; script-supporting elements p159	globals p162	HTMLTableRowElement p510
track p427	Timed text track	none	audio p425; video p420	empty	globals p162; default p429; kind p428; label p429; src p428; srclang p428	HTMLTrackElement p428
u p304	Unarticulated annotation	flow p157; phrasing p157; palpable p158	phrasing p157	phrasing p157	globals p162	HTMLElement p149
ul p249	List	flow p157; palpable p158*	flow p157	li p251; script-supporting elements p159	globals p162	HTMLUListElement p249
var p298	Variable	flow p157; phrasing p157; palpable p158	phrasing p157	phrasing p157	globals p162	HTMLElement p149
video p420	Video player	flow p157; phrasing p157; embedded p158; interactive p158; palpable p158	phrasing p157	source p355*; track p427*; transparent p159*	globals p162; src p433; crossorigin p433; poster p421; preload p446; autoplay p452; playsinline p422; loading p431; loop p450; muted p483; controls p481; width p495; height p495	HTMLVideoElement p421
wbr p311	Line breaking opportunity	flow p157; phrasing p157	phrasing p157	empty	globals p162	HTMLElement p149
autonomous custom elements p796	Author-defined elements	flow p157; phrasing p157; palpable p158	flow p157; phrasing p157	transparent p159	globals p162; any, as decided by the element's author	Supplied by the element's author (inherits from HTMLElement p149)

An asterisk (*) in a cell indicates that the actual rules are more complicated than indicated in the table above.

† Categories in the "Parents" column refer to parents that list the given categories in their content model, not to elements that themselves are in those categories. For example, the a p267 element's "Parents" column says "phrasing", so any element whose content model contains the "phrasing" category could be a parent of an a p267 element. Since the "flow" category includes all the "phrasing" elements, that means the th p513 element could be a parent to an a p267 element.

Element content categories §[p15 61]

This section is non-normative.

List of element content categories

Category	Elements	Elements with exceptions
Metadata content[p156]	base[p182]; link[p184]; meta[p196]; noscript[p703]; script[p685]; style[p207]; template[p705]; title[p181]	—
Flow content[p157]	a[p267]; abbr[p279]; address[p230]; article[p214]; aside[p222]; audio[p425]; b[p303]; bdi[p307]; bdo[p309]; blockquote[p244]; br[p310]; button[p585]; canvas[p713]; cite[p275]; code[p297]; data[p289]; datalist[p598]; del[p351]; details[p666]; dfn[p278]; dialog[p675]; div[p266]; dl[p253]; em[p270]; embed[p413]; fieldset[p620]; figure[p259]; footer[p227]; form[p532]; h1[p224]; h2[p224]; h3[p224]; h4[p224]; h5[p224]; h6[p224]; header[p226]; hgroup[p225]; hr[p241]; i[p302]; iframe[p404]; img[p359]; input[p539]; ins[p350]; kbd[p300]; label[p536]; map[p487]; mark[p305]; MathML math; menu[p250]; meter[p616]; nav[p219]; noscript[p703]; object[p416]; ol[p247]; output[p611]; p[p238]; picture[p355]; pre[p242]; progress[p614]; q[p277]; ruby[p281]; s[p274]; samp[p299]; script[p685]; search[p264]; section[p216]; select[p590]; slot[p712]; small[p272]; span[p309]; strong[p271]; sub[p301]; sup[p301]; SVG svg; table[p496]; template[p705]; textarea[p606]; time[p290]; u[p304]; ul[p249]; var[p298]; video[p420]; wbr[p311]; autonomous custom elements[p796]; Text[p158]	area[p489] (if it is a descendant of a map[p487] element); link[p184] (if it is allowed in the body[p186]); main[p262] (if it is a hierarchically correct main element[p263]); meta[p196] (if the itemprop[p833] attribute is present)
Sectioning content[p157]	article[p214]; aside[p222]; nav[p219]; section[p216]	—
Heading content[p157]	h1[p224]; h2[p224]; h3[p224]; h4[p224]; h5[p224]; h6[p224]; hgroup[p225]	—
Phrasing content[p157]	a[p267]; abbr[p279]; audio[p425]; b[p303]; bdi[p307]; bdo[p309]; br[p310]; button[p585]; canvas[p713]; cite[p275]; code[p297]; data[p289]; datalist[p598]; del[p351]; dfn[p278]; em[p270]; embed[p413]; i[p302]; iframe[p404]; img[p359]; input[p539]; ins[p350]; kbd[p300]; label[p536]; map[p487]; mark[p305]; MathML math; meter[p616]; noscript[p703]; object[p416]; output[p611]; picture[p355]; progress[p614]; q[p277]; ruby[p281]; s[p274]; samp[p299]; script[p685]; select[p590]; selectedcontent[p624]; slot[p712]; small[p272]; span[p309]; strong[p271]; sub[p301]; sup[p301]; SVG svg; template[p705]; textarea[p606]; time[p290]; u[p304]; var[p298]; video[p420]; wbr[p311]; autonomous custom elements[p796]; Text[p158]	area[p489] (if it is a descendant of a map[p487] element); link[p184] (if it is allowed in the body[p186]); meta[p196] (if the itemprop[p833] attribute is present)
Embedded content[p158]	audio[p425]; canvas[p713]; embed[p413]; iframe[p404]; img[p359]; MathML math; object[p416]; picture[p355]; SVG svg; video[p420]	—
Interactive content[p158]	button[p585]; details[p666]; embed[p413]; iframe[p404]; label[p536]; select[p590]; textarea[p606]	a[p267] (if the href[p314] attribute is present); audio[p425] (if the controls[p481] attribute is present); img[p359] (if the usemap[p491] or controls[p363] attribute is present); input[p539] (if the type[p541] attribute is *not* in the Hidden[p546] state); video[p420] (if the controls[p481] attribute is present)
Form-associated elements[p532]	button[p585]; fieldset[p620]; input[p539]; label[p536]; object[p416]; output[p611]; select[p590]; textarea[p606]; img[p359]; form-associated custom elements[p797]	—
Listed elements[p532]	button[p585]; fieldset[p620]; input[p539]; object[p416]; output[p611]; select[p590]; textarea[p606]; form-associated custom elements[p797]	—
Submittable elements[p532]	button[p585]; input[p539]; select[p590]; textarea[p606]; form-associated custom elements[p797]	—
Resettable elements[p532]	input[p539]; output[p611]; select[p590]; textarea[p606]; form-associated custom elements[p797]	—
Autocapitalize-and-autocorrect inheriting elements[p532]	button[p585]; fieldset[p620]; input[p539]; output[p611]; select[p590]; textarea[p606]	—
Labelable elements[p532]	button[p585]; input[p539]; meter[p616]; output[p611]; progress[p614]; select[p590]; textarea[p606]; form-associated custom elements[p797]	—
Palpable content[p158]	a[p267]; abbr[p279]; address[p230]; article[p214]; aside[p222]; b[p303]; bdi[p307]; bdo[p309]; blockquote[p244]; button[p585]; canvas[p713]; cite[p275]; code[p297]; data[p289]; del[p351]; details[p666]; dfn[p278]; div[p266]; em[p270]; embed[p413]; fieldset[p620]; figure[p259]; footer[p227]; form[p532]; h1[p224]; h2[p224]; h3[p224]; h4[p224]; h5[p224]; h6[p224]; header[p226]; hgroup[p225]; i[p302]; iframe[p404]; img[p359]; ins[p350]; kbd[p300]; label[p536]; main[p262]; map[p487]; mark[p305]; MathML math; meter[p616]; nav[p219]; object[p416]; output[p611]; p[p238]; picture[p355]; pre[p242]; progress[p614]; q[p277]; ruby[p281]; s[p274]; samp[p299]; search[p264]; section[p216]; select[p590]; small[p272]; span[p309]; strong[p271]; sub[p301]; sup[p301]; SVG svg; table[p496]; textarea[p606]; time[p290]; u[p304]; var[p298]; video[p420]; autonomous custom elements[p796]	audio[p425] (if the controls[p481] attribute is present); dl[p253] (if the element's children include at least one name-value group); input[p539] (if the type[p541] attribute is *not* in the Hidden[p546] state); menu[p250] (if the element's children include at least one li[p251] element); ol[p247] (if the element's children include at least one li[p251] element); ul[p249] (if the element's children include at least one li[p251] element); Text[p158] that is not inter-element whitespace[p155]
Script-supporting elements[p159]	script[p685]; template[p705]	—

Attributes §[p15] 62

This section is non-normative.

List of attributes (excluding event handler content attributes)

Attribute	Element(s)	Description	Value
abbr	th[p514]	Alternative label to use for the header cell when referencing the cell in other contexts	Text[p154]*
accept	input[p563]	Hint for expected file type in file upload controls[p563]	Set of comma-separated tokens[p99]* consisting of valid MIME type strings with no parameters or audio/*, video/*, or image/*
accept-charset	form[p533]	Character encodings to use for form submission[p657]	ASCII case-insensitive match for "UTF-8"
accesskey	HTML elements[p890]	Keyboard shortcut to activate or focus element	Ordered set of unique space-separated tokens[p99], none of which are identical to another, each consisting of one code point in length
action	form[p631]	URL to use for form submission[p657]	Valid non-empty URL potentially surrounded by spaces[p100]
allow	iframe[p411]	Permissions policy to be applied to the iframe[p404]'s contents	Serialized permissions policy
allowfullscreen	iframe[p411]	Whether to allow the iframe[p404]'s contents to use requestFullscreen()	Boolean attribute[p79]
alpha	input[p559]	Allow the color's alpha component to be set	Boolean attribute[p79]
alt	area[p490]; img[p360]; input[p567]	Replacement text for use when images are not available	Text[p154]*
as	link[p188]	Destination for a preload request (for rel[p185]="preload[p340]" and rel[p185]="modulepreload[p335]")	Preload destination[p342], for rel[p185]="preload[p340]"; module preload destination[p335], for rel[p185]="modulepreload[p335]"
async	script[p687]	Execute script when available, without blocking while fetching	Boolean attribute[p79]
autocapitalize	HTML elements[p898]	Recommended autocapitalization behavior (for supported input methods)	"on[p898]"; "off[p898]"; "none[p898]"; "sentences[p898]"; "words[p898]"; "characters[p898]"
autocomplete	form[p533]	Default setting for autofill feature for controls in the form	"on"; "off"
autocomplete	input[p633]; select[p633]; textarea[p633]	Hint for form autofill feature	Autofill field[p636] name and related tokens*
autocorrect	HTML elements[p899]	Recommended autocorrection behavior (for supported input methods)	"on[p899]"; "off[p899]"; the empty string
autofocus	HTML elements[p887]	Automatically focus the element when the page is loaded	Boolean attribute[p79]
autoplay	audio[p452]; video[p452]	Hint that the media resource[p432] can be started automatically when the page is loaded	Boolean attribute[p79]
blocking	link[p188]; script[p688]; style[p209]	Whether the element is potentially render-blocking[p107]	Unordered set of unique space-separated tokens[p99]*
charset	meta[p197]	Character encoding declaration[p206]	"utf-8"
checked	input[p543]	Whether the control is checked	Boolean attribute[p79]
cite	blockquote[p245]; del[p352]; ins[p352]; q[p277]	Link to the source of the quotation or more information about the edit	Valid URL potentially surrounded by spaces[p100]
class	HTML elements[p162]	Classes to which the element belongs	Set of space-separated tokens[p99]
closedby	dialog[p677]	Which user actions will close the dialog	"any[p677]"; "closerequest[p677]"; "none[p677]";
color	link[p188]	Color to use when customizing a site's icon (for rel[p185]="mask-icon")	CSS <color>
colorspace	input[p560]	The color space of the serialized color	"limited-srgb[p560]"; "display-p3[p560]"

Attribute	Element(s)	Description	Value
cols	textarea[p609]	Maximum number of characters per line	Valid non-negative integer[p81] greater than zero
colspan	td[p515]; th[p515]	Number of columns that the cell is to span	Valid non-negative integer[p81] greater than zero
command	button[p586]	Indicates to the targeted element which action to take.	"toggle-popover[p586]"; "show-popover[p586]"; "hide-popover[p586]"; "close[p586]"; "request-close[p586]"; "show-modal[p586]"; a custom command keyword[p586]
commandfor	button[p586]	Targets another element to be invoked.	ID*
content	meta[p197]	Value of the element	Text[p154]*
contenteditable	HTML elements[p892]	Whether the element is editable	"true"; "false"; "plaintext-only"; the empty string
controls	audio[p481]; video[p481] img[p363];	Show user agent controls	Boolean attribute[p79]
coords	area[p490]	Coordinates for the shape to be created in an image map[p491]	Valid list of floating-point numbers[p84]*
crossorigin	audio[p433]; img[p361]; link[p186]; script[p688]; video[p433]	How the element handles crossorigin requests	"anonymous[p103]"; "use-credentials[p103]"; the empty string
data	object[p417]	Address of the resource	Valid non-empty URL potentially surrounded by spaces[p100]
datetime	del[p352]; ins[p352]	Date and (optionally) time of the change	Valid date string with optional time[p97]
datetime	time[p290]	Machine-readable value	Valid month string[p86], valid date string[p87], valid yearless date string[p88], valid time string[p88], valid local date and time string[p89], valid time-zone offset string[p90], valid global date and time string[p92], valid week string[p93], valid non-negative integer[p81], or valid duration string[p94]
decoding	img[p361]	Decoding hint to use when processing this image for presentation	"sync[p380]"; "async[p380]"; "auto[p380]"
default	track[p429]	Enable the track if no other text track[p466] is more suitable	Boolean attribute[p79]
defer	script[p687]	Defer script execution	Boolean attribute[p79]
dir	HTML elements[p167]	The text directionality[p168] of the element	"ltr[p167]"; "rtl[p167]"; "auto[p167]"
dir	bdo[p309]	The text directionality[p168] of the element	"ltr[p167]"; "rtl[p167]"
dirname	input[p629]; textarea[p629]	Name of form control to use for sending the element's directionality[p168] in form submission[p657]	Text[p154]*
disabled	button[p630]; input[p630]; optgroup[p600]; option[p602]; select[p630]; textarea[p630]; form-associated custom elements[p630]	Whether the form control is disabled	Boolean attribute[p79]
disabled	fieldset[p621]	Whether the descendant form controls, except any inside legend[p623], are disabled	Boolean attribute[p79]
disabled	link[p188]	Whether the link is disabled	Boolean attribute[p79]
download	a[p314]; area[p314]	Whether to download the resource instead of navigating to it, and its filename if so	Text
draggable	HTML elements[p924]	Whether the element is draggable	"true"; "false"
enctype	form[p632]	Entry list[p661] encoding type to use for form submission[p657]	"application/x-www-form-urlencoded[p632]"; "multipart/form-data[p632]"; "text/plain[p632]"
enterkeyhint	HTML elements[p901]	Hint for selecting an enter key action	"enter[p901]"; "done[p901]"; "go[p901]"; "next[p901]"; "previous[p901]"; "search[p901]"; "send[p901]"
fetchpriority	img[p361]; link[p189]; script[p688]	Sets the priority for fetches initiated by the element	"auto[p108]"; "high[p108]"; "low[p108]"
for	label[p537]	Associate the label with form control	ID*
for	output[p612]	Specifies controls from which the output was calculated	Unordered set of unique space-separated tokens[p99] consisting of IDs*

Attribute	Element(s)	Description	Value
for	template[p706]	Updates existing content	Text
form	button[p627]; fieldset[p627]; input[p627]; object[p627]; output[p627]; select[p627]; textarea[p627]; form-associated custom elements[p627]	Associates the element with a form[p532] element	ID*
formaction	button[p631]; input[p631]	URL to use for form submission[p657]	Valid non-empty URL potentially surrounded by spaces[p100]
formenctype	button[p632]; input[p632]	Entry list[p661] encoding type to use for form submission[p657]	"application/x-www-form-urlencoded[p632]"; "multipart/form-data[p632]"; "text/plain[p632]"
formmethod	button[p631]; input[p631]	Variant to use for form submission[p657]	"get[p631]"; "post[p631]"; "dialog[p631]"
formnovalidate	button[p632]; input[p632]	Bypass form control validation for form submission[p657]	Boolean attribute[p79]
formtarget	button[p632]; input[p632]	Navigable[p1035] for form submission[p657]	Valid navigable target name or keyword[p1043]
headers	td[p515]; th[p515]	The header cells for this cell	Unordered set of unique space-separated tokens[p99] consisting of IDs*
headingoffset	HTML elements[p878]	Offsets heading levels for descendants	Valid non-negative integer[p81] between 0 and 8
headingreset	HTML elements[p878]	Prevents a heading offset computation from traversing beyond the element with the attribute	Boolean attribute[p79]
height	canvas[p715]; embed[p495]; iframe[p495]; img[p495]; input[p495]; object[p495]; source[p495] (in picture[p355]); video[p495]	Vertical dimension	Valid non-negative integer[p81]
hidden	HTML elements[p862]	Whether the element is relevant	"until-found[p862]"; "hidden[p862]"; the empty string
high	meter[p617]	Low limit of high range	Valid floating-point number[p81]*
href	a[p314]; area[p314]	Address of the hyperlink[p313]	Valid URL potentially surrounded by spaces[p100]
href	link[p185]	Address of the hyperlink[p313]	Valid non-empty URL potentially surrounded by spaces[p100]
href	base[p183]	Document base URL[p101]	Valid URL potentially surrounded by spaces[p100]
hreflang	a[p316]; link[p186]	Language of the linked resource	Valid BCP 47 language tag
http-equiv	meta[p202]	Pragma directive	"content-type[p203]"; "default-style[p203]"; "refresh[p203]"; "x-ua-compatible[p203]"; "content-security-policy[p203]"
id	HTML elements[p162]	The element's ID	Text[p154]*
imagesizes	link[p187]	Image sizes for different page layouts (for rel[p185]="preload[p340]")	Valid source size list[p376]
imagesrcset	link[p187]	Images to use in different situations, e.g., high-resolution displays, small monitors, etc. (for rel[p185]="preload[p340]")	Comma-separated list of image candidate strings[p375]
inert	HTML elements[p866]	Whether the element is inert[p866].	Boolean attribute[p79]
inputmode	HTML elements[p900]	Hint for selecting an input modality	"none[p900]"; "text[p900]"; "tel[p900]"; "email[p900]"; "url[p900]"; "numeric[p900]"; "decimal[p900]"; "search[p900]"
integrity	link[p186]; script[p688]	Integrity metadata used in *Subresource Integrity* checks [SRI][p1586]	Text[p154]
is	HTML elements[p796]	Creates a customized built-in element[p796]	Valid custom element name[p797] of a defined customized built-in element[p796]
ismap	img[p363]	Whether the image is a server-side image map	Boolean attribute[p79]
itemid	HTML elements[p832]	Global identifier[p832] for a microdata item	Valid URL potentially surrounded by spaces[p100]
itemprop	HTML elements[p833]	Property names[p834] of a microdata item	Unordered set of unique space-separated tokens[p99] consisting of valid absolute URLs, defined property names[p834], or text*
itemref	HTML elements[p832]	Referenced[p148] elements	Unordered set of unique space-separated tokens[p99] consisting of IDs*

Attribute	Element(s)	Description	Value
itemscope	HTML elements[p831]	Introduces a microdata item	Boolean attribute[p79]
itemtype	HTML elements[p831]	Item types[p831] of a microdata item	Unordered set of unique space-separated tokens[p99] consisting of valid absolute URLs*
kind	track[p428]	The type of text track	"subtitles[p428]"; "captions[p428]"; "descriptions[p428]"; "chapters[p428]"; "metadata[p428]"
label	optgroup[p601]; option[p603]; track[p429]	User-visible label	Text[p154]
lang	HTML elements[p165]	Language[p166] of the element	Valid BCP 47 language tag or the empty string
list	input[p576]	List of autocomplete options	ID*
loading	iframe[p412]; img[p361]; audio[p431]; video[p431]	Used when determining loading deferral	"lazy[p105]"; "eager[p105]"
loop	audio[p450]; video[p450]	Whether to loop the media resource[p432]	Boolean attribute[p79]
low	meter[p617]	High limit of low range	Valid floating-point number[p81]*
max	input[p574]	Maximum value	Varies*
max	meter[p617]; progress[p614]	Upper bound of range	Valid floating-point number[p81]*
maxlength	input[p569]; textarea[p609]	Maximum length of value	Valid non-negative integer[p81]
media	link[p186]; meta[p197]; source[p356]; style[p208]	Applicable media	Valid media query list[p100]
method	form[p631]	Variant to use for form submission[p657]	"get[p631]"; "post[p631]"; "dialog[p631]"
min	input[p574]	Minimum value	Varies*
min	meter[p617]	Lower bound of range	Valid floating-point number[p81]*
minlength	input[p569]; textarea[p609]	Minimum length of value	Valid non-negative integer[p81]
multiple	input[p572]; select[p591]	Whether to allow multiple values	Boolean attribute[p79]
muted	audio[p483]; video[p483]	Whether to mute the media resource[p432] by default	Boolean attribute[p79]
name	button[p628]; fieldset[p628]; input[p628]; output[p628]; select[p628]; textarea[p628]; form-associated custom elements[p628]	Name of the element to use for form submission[p657] and in the form.elements[p534] API	Text[p154]*
name	details[p667]	Name of group of mutually-exclusive details[p666] elements	Text[p154]*
name	form[p533]	Name of form to use in the document.forms[p144] API	Text[p154]*
name	iframe[p409]; object[p417]	Name of content navigable[p1038]	Valid navigable target name or keyword[p1043]
name	map[p488]	Name of image map[p491] to reference[p148] from the usemap[p491] attribute	Text[p154]*
name	meta[p197]	Metadata name	Text[p154]*
name	slot[p712]	Name of shadow tree slot	Text[p154]
nomodule	script[p687]	Prevents execution in user agents that support module scripts[p1153]	Boolean attribute[p79]
nonce	HTML elements[p104]	Cryptographic nonce used in *Content Security Policy* checks [CSP][p1580]	Text[p154]
novalidate	form[p632]	Bypass form control validation for form submission[p657]	Boolean attribute[p79]
open	details[p667]	Whether the details are visible	Boolean attribute[p79]
open	dialog[p677]	Whether the dialog box is showing	Boolean attribute[p79]
optimum	meter[p617]	Optimum value in gauge	Valid floating-point number[p81]*
pattern	input[p573]	Pattern to be matched by the form control's value	Regular expression matching the JavaScript *Pattern* production
ping	a[p314]; area[p314]	URLs to ping	Set of space-separated tokens[p99] consisting of valid non-empty URLs[p100]
placeholder	input[p578]; textarea[p610]	User-visible label to be placed within the form control	Text[p154]*
playsinline	video[p422]	Encourage the user agent to display video content within the	Boolean attribute[p79]

Attribute	Element(s)	Description	Value
		element's playback area	
popover	HTML elements[p925]	Makes the element a popover[p925] element	"auto[p926]"; "manual[p926]"; "hint[p926]"; the empty string
popovertarget	button[p935]; input[p935]	Targets a popover element to toggle, show, or hide	ID*
popovertargetaction	button[p935]; input[p935]	Indicates whether a targeted popover element is to be toggled, shown, or hidden	"toggle[p935]"; "show[p935]"; "hide[p935]"
poster	video[p421]	Poster frame to show prior to video playback	Valid non-empty URL potentially surrounded by spaces[p100]
preload	audio[p446]; video[p446]	Hints how much buffering the media resource[p432] will likely need	"none[p446]"; "metadata[p446]"; "auto[p446]"; the empty string
readonly	input[p570]; textarea[p608]	Whether to allow the value to be edited by the user	Boolean attribute[p79]
readonly	form-associated custom elements[p797]	Affects willValidate[p654], plus any behavior added by the custom element author	Boolean attribute[p79]
referrerpolicy	a[p314]; area[p314]; iframe[p412]; img[p361]; link[p187]; script[p688]	Referrer policy for fetches initiated by the element	Referrer policy
rel	a[p314]; area[p314]	Relationship between the location in the document containing the hyperlink[p313] and the destination resource	Unordered set of unique space-separated tokens[p99]*
rel	link[p185]	Relationship between the document containing the hyperlink[p313] and the destination resource	Unordered set of unique space-separated tokens[p99]*
required	input[p571]; select[p591]; textarea[p610]	Whether the control is required for form submission[p657]	Boolean attribute[p79]
reversed	ol[p248]	Number the list backwards	Boolean attribute[p79]
rows	textarea[p609]	Number of lines to show	Valid non-negative integer[p81] greater than zero
rowspan	td[p515]; th[p515]	Number of rows that the cell is to span	Valid non-negative integer[p81]
sandbox	iframe[p409]	Security rules for nested content	Unordered set of unique space-separated tokens[p99], ASCII case-insensitive, consisting of • "allow-downloads[p959]" • "allow-forms[p959]" • "allow-modals[p959]" • "allow-orientation-lock[p959]" • "allow-pointer-lock[p959]" • "allow-popups[p958]" • "allow-popups-to-escape-sandbox[p959]" • "allow-presentation[p959]" • "allow-same-origin[p958]" • "allow-scripts[p959]" • "allow-top-navigation[p958]" • "allow-top-navigation-by-user-activation[p958]" • "allow-top-navigation-to-custom-protocols[p959]"
scope	th[p514]	Specifies which cells the header cell applies to	"row[p514]"; "col[p514]"; "rowgroup[p514]"; "colgroup[p514]"
selected	option[p603]	Whether the option is selected by default	Boolean attribute[p79]
shadowrootclonable	template[p706]	Sets clonable on a declarative shadow root	Boolean attribute[p79]
shadowrootcustomelementregistry	template[p706]	Enables declarative shadow roots to indicate they will use a custom element registry	Boolean attribute[p79]
shadowrootdelegatesfocus	template[p706]	Sets delegates focus on a declarative shadow root	Boolean attribute[p79]
shadowrootmode	template[p706]	Enables streaming declarative shadow roots	"open"; "closed"
shadowrootserializable	template[p706]	Sets serializable on a declarative shadow root	Boolean attribute[p79]
shadowrootslotassignment	template[p706]	Sets slot assignment on a	"named"; "manual"

Attribute	Element(s)	Description	Value
		declarative shadow root	
shape	area	The kind of shape to be created in an image map	"circle"; "default"; "poly"; "rect"
size	input; select	Size of the control	Valid non-negative integer greater than zero
sizes	link	Sizes of the icons (for rel="icon")	Unordered set of unique space-separated tokens, ASCII case-insensitive, consisting of sizes*
sizes	img; source	Image sizes for different page layouts	Valid source size list
slot	HTML elements	The element's desired slot	Text
span	col; colgroup	Number of columns spanned by the element	Valid non-negative integer greater than zero
spellcheck	HTML elements	Whether the element is to have its spelling and grammar checked	"true"; "false"; the empty string
src	audio; embed; iframe; img; input; script; source (in video or audio); track; video	Address of the resource	Valid non-empty URL potentially surrounded by spaces
srcdoc	iframe	A document to render in the iframe	The source of an iframe srcdoc document*
srclang	track	Language of the text track	Valid BCP 47 language tag
srcset	img; source	Images to use in different situations, e.g., high-resolution displays, small monitors, etc.	Comma-separated list of image candidate strings
start	ol	Starting value of the list	Valid integer
step	input	Granularity to be matched by the form control's value	Valid floating-point number greater than zero, or "any"
style	HTML elements	Presentational and formatting instructions	CSS declarations*
tabindex	HTML elements	Whether the element is focusable and sequentially focusable, and the relative order of the element for the purposes of sequential focus navigation	Valid integer
target	a; area	Navigable for hyperlink navigation	Valid navigable target name or keyword
target	base	Default navigable for hyperlink navigation and form submission	Valid navigable target name or keyword
target	form	Navigable for form submission	Valid navigable target name or keyword
title	HTML elements	Advisory information for the element	Text
title	abbr; dfn	Full term or expansion of abbreviation	Text
title	input	Description of pattern (when used with pattern attribute)	Text
title	link	Title of the link	Text
title	link; style	CSS style sheet set name	Text
translate	HTML elements	Whether the element is to be translated when the page is localized	"yes"; "no"; the empty string
type	a; link	Hint for the type of the referenced resource	Valid MIME type string
type	button	Type of button	"submit"; "reset"; "button"
type	embed; object; source	Type of embedded resource	Valid MIME type string
type	input	Type of form control	input type keyword
type	ol	Kind of list marker	"1"; "a"; "A"; "i"; "I"
type	script	Type of script	"module"; "importmap"; "speculationrules"; a valid MIME type string that is not a JavaScript MIME type essence

Attribute	Element(s)	Description	Value
			match
usemap	img	Name of image map to use	Valid hash-name reference*
value	button; option	Value to be used for form submission	Text
value	data	Machine-readable value	Text*
value	input	Value of the form control	Varies*
value	li	Ordinal value of the list item	Valid integer
value	meter; progress	Current value of the element	Valid floating-point number
width	canvas; embed; iframe; img; input; object; source (in picture); video	Horizontal dimension	Valid non-negative integer
wrap	textarea	How the value of the form control is to be wrapped for form submission	"soft"; "hard"
writingsuggestions	HTML elements	Whether the element can offer writing suggestions or not.	"true"; "false"; the empty string

An asterisk (*) in a cell indicates that the actual rules are more complicated than indicated in the table above.

List of event handler content attributes

Attribute	Element(s)	Description	Value
onafterprint	body	afterprint event handler for Window object	Event handler content attribute
onauxclick	HTML elements	auxclick event handler	Event handler content attribute
onbeforeinput	HTML elements	beforeinput event handler	Event handler content attribute
onbeforematch	HTML elements	beforematch event handler	Event handler content attribute
onbeforeprint	body	beforeprint event handler for Window object	Event handler content attribute
onbeforeunload	body	beforeunload event handler for Window object	Event handler content attribute
onbeforetoggle	HTML elements	beforetoggle event handler	Event handler content attribute
onblur	HTML elements	blur event handler	Event handler content attribute
oncancel	HTML elements	cancel event handler	Event handler content attribute
oncanplay	HTML elements	canplay event handler	Event handler content attribute
oncanplaythrough	HTML elements	canplaythrough event handler	Event handler content attribute
onchange	HTML elements	change event handler	Event handler content attribute
onclick	HTML elements	click event handler	Event handler content attribute
onclose	HTML elements	close event handler	Event handler content attribute
oncommand	HTML elements	command event handler	Event handler content attribute
oncontextlost	HTML elements	contextlost event handler	Event handler content attribute
oncontextmenu	HTML elements	contextmenu event handler	Event handler content attribute
oncontextrestored	HTML elements	contextrestored event handler	Event handler content attribute
oncopy	HTML elements	copy event handler	Event handler content attribute
oncuechange	HTML elements	cuechange event handler	Event handler content attribute
oncut	HTML elements	cut event handler	Event handler content attribute
ondblclick	HTML elements	dblclick event handler	Event handler content attribute
ondrag	HTML elements	drag event handler	Event handler content attribute
ondragend	HTML elements	dragend event handler	Event handler content attribute
ondragenter	HTML elements	dragenter event handler	Event handler content attribute
ondragleave	HTML elements	dragleave event handler	Event handler content attribute
ondragover	HTML elements	dragover event handler	Event handler content attribute
ondragstart	HTML elements	dragstart event handler	Event handler content attribute
ondrop	HTML elements	drop event handler	Event handler content attribute
ondurationchange	HTML elements	durationchange event handler	Event handler content attribute
onemptied	HTML elements	emptied event handler	Event handler content attribute
onended	HTML elements	ended event handler	Event handler content attribute
onerror	HTML elements	error event handler	Event handler content attribute

Attribute	Element(s)	Description	Value
onfocus	HTML elements p1215	focus p1575 event handler	Event handler content attribute p1207
onformdata	HTML elements p1214	formdata p1575 event handler	Event handler content attribute p1207
onhashchange	body p1215	hashchange p1575 event handler for Window p965 object	Event handler content attribute p1207
oninput	HTML elements p1214	input event handler	Event handler content attribute p1207
oninvalid	HTML elements p1214	invalid p1575 event handler	Event handler content attribute p1207
onkeydown	HTML elements p1214	keydown event handler	Event handler content attribute p1207
onkeypress	HTML elements p1214	keypress event handler	Event handler content attribute p1207
onkeyup	HTML elements p1214	keyup event handler	Event handler content attribute p1207
onlanguagechange	body p1215	languagechange p1575 event handler for Window p965 object	Event handler content attribute p1207
onload	HTML elements p1215	load p1575 event handler	Event handler content attribute p1207
onloadeddata	HTML elements p1214	loadeddata p485 event handler	Event handler content attribute p1207
onloadedmetadata	HTML elements p1214	loadedmetadata p485 event handler	Event handler content attribute p1207
onloadstart	HTML elements p1214	loadstart p484 event handler	Event handler content attribute p1207
onmessage	body p1215	message p1575 event handler for Window p965 object	Event handler content attribute p1207
onmessageerror	body p1215	messageerror p1575 event handler for Window p965 object	Event handler content attribute p1207
onmousedown	HTML elements p1214	mousedown event handler	Event handler content attribute p1207
onmouseenter	HTML elements p1214	mouseenter event handler	Event handler content attribute p1207
onmouseleave	HTML elements p1214	mouseleave event handler	Event handler content attribute p1207
onmousemove	HTML elements p1214	mousemove event handler	Event handler content attribute p1207
onmouseout	HTML elements p1214	mouseout event handler	Event handler content attribute p1207
onmouseover	HTML elements p1214	mouseover event handler	Event handler content attribute p1207
onmouseup	HTML elements p1214	mouseup event handler	Event handler content attribute p1207
onoffline	body p1215	offline p1576 event handler for Window p965 object	Event handler content attribute p1207
ononline	body p1215	online p1576 event handler for Window p965 object	Event handler content attribute p1207
onpagehide	body p1215	pagehide p1576 event handler for Window p965 object	Event handler content attribute p1207
onpagereveal	body p1215	pagereveal p1576 event handler for Window p965 object	Event handler content attribute p1207
onpageshow	body p1215	pageshow p1576 event handler for Window p965 object	Event handler content attribute p1207
onpageswap	body p1215	pageswap p1576 event handler for Window p965 object	Event handler content attribute p1207
onpaste	HTML elements p1214	paste event handler	Event handler content attribute p1207
onpause	HTML elements p1214	pause p485 event handler	Event handler content attribute p1207
onplay	HTML elements p1214	play p485 event handler	Event handler content attribute p1207
onplaying	HTML elements p1214	playing p485 event handler	Event handler content attribute p1207
onpopstate	body p1215	popstate p1576 event handler for Window p965 object	Event handler content attribute p1207
onprogress	HTML elements p1214	progress p484 event handler	Event handler content attribute p1207
onratechange	HTML elements p1214	ratechange p485 event handler	Event handler content attribute p1207
onreset	HTML elements p1214	reset p1576 event handler	Event handler content attribute p1207
onresize	HTML elements p1215	resize event handler	Event handler content attribute p1207
onrejectionhandled	body p1215	rejectionhandled p1576 event handler for Window p965 object	Event handler content attribute p1207
onscroll	HTML elements p1215	scroll event handler	Event handler content attribute p1207
onscrollend	HTML elements p1214	scrollend event handler	Event handler content attribute p1207
onsecuritypolicyviolation	HTML elements p1214	securitypolicyviolation event handler	Event handler content attribute p1207
onseeked	HTML elements p1214	seeked p485 event handler	Event handler content attribute p1207
onseeking	HTML elements p1214	seeking p485 event handler	Event handler content attribute p1207
onselect	HTML elements p1214	select p1576 event handler	Event handler content attribute p1207
onslotchange	HTML elements p1214	slotchange event handler	Event handler content attribute p1207
onstalled	HTML elements p1214	stalled p485 event handler	Event handler content attribute p1207
onstorage	body p1215	storage p1576 event handler for Window p965 object	Event handler content attribute p1207
onsubmit	HTML elements p1214	submit p1576 event handler	Event handler content attribute p1207
onsuspend	HTML elements p1214	suspend p484 event handler	Event handler content attribute p1207
ontimeupdate	HTML elements p1214	timeupdate p485 event handler	Event handler content attribute p1207
ontoggle	HTML elements p1214	toggle p1576 event handler	Event handler content attribute p1207
onunhandledrejection	body p1215	unhandledrejection p1576 event handler for Window p965 object	Event handler content attribute p1207
onunload	body p1215	unload p1576 event handler for Window p965 object	Event handler content attribute p1207
onvolumechange	HTML elements p1214	volumechange p486 event handler	Event handler content attribute p1207

Attribute	Element(s)	Description	Value
onwaiting	HTML elements p1214	waiting p485 event handler	Event handler content attribute p1207
onwheel	HTML elements p1215	wheel event handler	Event handler content attribute p1207

Element interfaces §p15 70

This section is non-normative.

List of interfaces for elements

Element(s)	Interface(s)
a p267	HTMLAnchorElement p268 : HTMLElement p149
abbr p279	HTMLElement p149
address p230	HTMLElement p149
area p489	HTMLAreaElement p489 : HTMLElement p149
article p214	HTMLElement p149
aside p222	HTMLElement p149
audio p425	HTMLAudioElement p426 : HTMLMediaElement p430 : HTMLElement p149
b p303	HTMLElement p149
base p182	HTMLBaseElement p182 : HTMLElement p149
bdi p307	HTMLElement p149
bdo p309	HTMLElement p149
blockquote p244	HTMLQuoteElement p244 : HTMLElement p149
body p212	HTMLBodyElement p213 : HTMLElement p149
br p310	HTMLBRElement p310 : HTMLElement p149
button p585	HTMLButtonElement p585 : HTMLElement p149
canvas p713	HTMLCanvasElement p714 : HTMLElement p149
caption p504	HTMLTableCaptionElement p504 : HTMLElement p149
cite p275	HTMLElement p149
code p297	HTMLElement p149
col p506	HTMLTableColElement p506 : HTMLElement p149
colgroup p505	HTMLTableColElement p506 : HTMLElement p149
data p289	HTMLDataElement p289 : HTMLElement p149
datalist p598	HTMLDataListElement p599 : HTMLElement p149
dd p258	HTMLElement p149
del p351	HTMLModElement p352 : HTMLElement p149
details p666	HTMLDetailsElement p666 : HTMLElement p149
dfn p278	HTMLElement p149
dialog p675	HTMLDialogElement p676 : HTMLElement p149
div p266	HTMLDivElement p266 : HTMLElement p149
dl p253	HTMLDListElement p253 : HTMLElement p149
dt p257	HTMLElement p149
em p270	HTMLElement p149
embed p413	HTMLEmbedElement p413 : HTMLElement p149
fieldset p620	HTMLFieldSetElement p621 : HTMLElement p149
figcaption p262	HTMLElement p149
figure p259	HTMLElement p149
footer p227	HTMLElement p149
form p532	HTMLFormElement p533 : HTMLElement p149
h1 p224	HTMLHeadingElement p224 : HTMLElement p149
h2 p224	HTMLHeadingElement p224 : HTMLElement p149
h3 p224	HTMLHeadingElement p224 : HTMLElement p149
h4 p224	HTMLHeadingElement p224 : HTMLElement p149
h5 p224	HTMLHeadingElement p224 : HTMLElement p149
h6 p224	HTMLHeadingElement p224 : HTMLElement p149

Element(s)	Interface(s)
head	HTMLHeadElement : HTMLElement
header	HTMLElement
hgroup	HTMLElement
hr	HTMLHRElement : HTMLElement
html	HTMLHtmlElement : HTMLElement
i	HTMLElement
iframe	HTMLIFrameElement : HTMLElement
img	HTMLImageElement : HTMLElement
input	HTMLInputElement : HTMLElement
ins	HTMLModElement : HTMLElement
kbd	HTMLElement
label	HTMLLabelElement : HTMLElement
legend	HTMLLegendElement : HTMLElement
li	HTMLLIElement : HTMLElement
link	HTMLLinkElement : HTMLElement
main	HTMLElement
map	HTMLMapElement : HTMLElement
mark	HTMLElement
menu	HTMLMenuElement : HTMLElement
meta	HTMLMetaElement : HTMLElement
meter	HTMLMeterElement : HTMLElement
nav	HTMLElement
noscript	HTMLElement
object	HTMLObjectElement : HTMLElement
ol	HTMLOListElement : HTMLElement
optgroup	HTMLOptGroupElement : HTMLElement
option	HTMLOptionElement : HTMLElement
output	HTMLOutputElement : HTMLElement
p	HTMLParagraphElement : HTMLElement
picture	HTMLPictureElement : HTMLElement
pre	HTMLPreElement : HTMLElement
progress	HTMLProgressElement : HTMLElement
q	HTMLQuoteElement : HTMLElement
rp	HTMLElement
rt	HTMLElement
ruby	HTMLElement
s	HTMLElement
samp	HTMLElement
search	HTMLElement
script	HTMLScriptElement : HTMLElement
section	HTMLElement
select	HTMLSelectElement : HTMLElement
selectedcontent	HTMLSelectedContentElement : HTMLElement
slot	HTMLSlotElement : HTMLElement
small	HTMLElement
source	HTMLSourceElement : HTMLElement
span	HTMLSpanElement : HTMLElement
strong	HTMLElement
style	HTMLStyleElement : HTMLElement
sub	HTMLElement
summary	HTMLElement
sup	HTMLElement
table	HTMLTableElement : HTMLElement
tbody	HTMLTableSectionElement : HTMLElement

Element(s)	Interface(s)
td p512	HTMLTableCellElement p512 : HTMLElement p149
template p705	HTMLTemplateElement p705 : HTMLElement p149
textarea p606	HTMLTextAreaElement p607 : HTMLElement p149
tfoot p509	HTMLTableSectionElement p507 : HTMLElement p149
th p513	HTMLTableCellElement p512 : HTMLElement p149
thead p508	HTMLTableSectionElement p507 : HTMLElement p149
time p290	HTMLTimeElement p290 : HTMLElement p149
title p181	HTMLTitleElement p181 : HTMLElement p149
tr p510	HTMLTableRowElement p510 : HTMLElement p149
track p427	HTMLTrackElement p428 : HTMLElement p149
u p304	HTMLElement p149
ul p249	HTMLUListElement p249 : HTMLElement p149
var p298	HTMLElement p149
video p420	HTMLVideoElement p421 : HTMLMediaElement p430 : HTMLElement p149
wbr p311	HTMLElement p149
custom elements p796	supplied by the element's author (inherits from HTMLElement p149)

All interfaces §p15 72

This section is non-normative.

- AudioTrack p462
- AudioTrackList p462
- BarProp p975
- BeforeUnloadEvent p1030
- BroadcastChannel p1303
- CanvasGradient p722
- CanvasPattern p722
- CanvasRenderingContext2D p719
- CloseWatcher p906
- CommandEvent p872
- CustomElementRegistry p799
- CustomStateSet p813
- DOMParser p1225
- DOMStringList p121
- DOMStringMap p173
- DataTransfer p911
- DataTransferItem p916
- DataTransferItemList p915
- DedicatedWorkerGlobalScope p1324
- Document p135, partial 1 p135 2 p1544
- DragEvent p917
- Element, partial p1224
- ElementInternals p809
- ErrorEvent p1168
- EventSource p1285
- External p1545
- FormDataEvent p665
- HTMLAllCollection p116
- HTMLAnchorElement p268, partial p1538
- HTMLAreaElement p489, partial p1538
- HTMLAudioElement p426
- HTMLBRElement p310, partial p1539
- HTMLBaseElement p182
- HTMLBodyElement p213, partial p1538
- HTMLButtonElement p585
- HTMLCanvasElement p714
- HTMLDListElement p253, partial p1539
- HTMLDataElement p289
- HTMLDataListElement p599
- HTMLDetailsElement p666
- HTMLDialogElement p676
- HTMLDirectoryElement p1539
- HTMLDivElement p266, partial p1539

- HTMLElement p149
- HTMLEmbedElement p413, partial p1539
- HTMLFieldSetElement p621
- HTMLFontElement p1540
- HTMLFormControlsCollection p117
- HTMLFormElement p533
- HTMLFrameElement p1538
- HTMLFrameSetElement p1537
- HTMLHRElement p241, partial p1540
- HTMLHeadElement p180
- HTMLHeadingElement p224, partial p1540
- HTMLHtmlElement p179, partial p1540
- HTMLIFrameElement p405, partial p1540
- HTMLImageElement p360, partial p1540
- HTMLInputElement p540, partial p1541
- HTMLLIElement p251, partial p1541
- HTMLLabelElement p537
- HTMLLegendElement p624, partial p1541
- HTMLLinkElement p185, partial p1541
- HTMLMapElement p488
- HTMLMarqueeElement p1535
- HTMLMediaElement p430
- HTMLMenuElement p250, partial p1541
- HTMLMetaElement p197, partial p1541
- HTMLMeterElement p616
- HTMLModElement p352
- HTMLOListElement p247, partial p1542
- HTMLObjectElement p416, partial p1542
- HTMLOptGroupElement p600
- HTMLOptionElement p602
- HTMLOptionsCollection p119
- HTMLOutputElement p612
- HTMLParagraphElement p238, partial p1542
- HTMLParamElement p1542
- HTMLPictureElement p355
- HTMLPreElement p243, partial p1542
- HTMLProgressElement p614
- HTMLQuoteElement p244
- HTMLScriptElement p686, partial p1543
- HTMLSelectElement p590
- HTMLSelectedContentElement p624
- HTMLSlotElement p712
- HTMLSourceElement p356
- HTMLSpanElement p310
- HTMLStyleElement p208, partial p1542
- HTMLTableCaptionElement p504, partial p1539
- HTMLTableCellElement p512, partial p1543
- HTMLTableColElement p506, partial p1539
- HTMLTableElement p496, partial p1543
- HTMLTableRowElement p510, partial p1543
- HTMLTableSectionElement p507, partial p1543
- HTMLTemplateElement p705
- HTMLTextAreaElement p607
- HTMLTimeElement p290
- HTMLTitleElement p181
- HTMLTrackElement p428
- HTMLUListElement p249, partial p1543
- HTMLUnknownElement p150
- HTMLVideoElement p421
- HashChangeEvent p1027
- History p987
- ImageBitmap p1275
- ImageBitmapRenderingContext p775
- ImageData p1272
- Location p980
- MediaError p432
- MessageChannel p1298
- MessageEvent p1283
- MessagePort p1299
- MimeType p1270
- MimeTypeArray p1269
- NavigateEvent p1013
- Navigation p995
- NavigationActivation p1012
- NavigationCurrentEntryChangeEvent p1026
- NavigationDestination p1018

- NavigationHistoryEntry[p999]
- NavigationPrecommitController[p1017]
- NavigationTransition[p1011]
- Navigator[p1261], partial[p870]
- NotRestoredReasonDetails[p1030]
- NotRestoredReasons[p1030]
- OffscreenCanvas[p777]
- OffscreenCanvasRenderingContext2D[p781]
- Origin[p939]
- PageRevealEvent[p1029]
- PageSwapEvent[p1028]
- PageTransitionEvent[p1029]
- Path2D[p723]
- Plugin[p48]
- PluginArray[p1269]
- PopStateEvent[p1027]
- PromiseRejectionEvent[p1169]
- RadioNodeList[p118]
- Range, partial[p1232]
- Sanitizer[p1233]
- ShadowRoot, partial[p1224]
- SharedWorker[p1333]
- SharedWorkerGlobalScope[p1324]
- Storage[p1349]
- StorageEvent[p1352]
- SubmitEvent[p665]
- TextMetrics[p722]
- TextTrack[p474]
- TextTrackCue[p478]
- TextTrackCueList[p477]
- TextTrackList[p474]
- TimeRanges[p483]
- ToggleEvent[p871]
- TrackEvent[p484]
- UserActivation[p870]
- ValidityState[p655]
- VideoTrack[p462]
- VideoTrackList[p462]
- VisibilityStateEntry[p865]
- Window[p965], partial[p1544]
- Worker[p1332]
- WorkerGlobalScope[p1322]
- WorkerLocation[p1337]
- WorkerNavigator[p1337]
- Worklet[p1345]
- WorkletGlobalScope[p1342]
- XMLSerializer[p1232]

Events §[p1574]

This section is non-normative.

The following table lists events fired by this document, excluding those already defined in media element events[p484] and drag-and-drop events[p923].

List of events

Event	Interface	Interesting targets	Description
DOMContentLoaded	Event	Document[p135]	Fired at the Document[p135] once the parser has finished
afterprint	Event	Window[p965]	Fired at the Window[p965] after printing
beforeprint	Event	Window[p965]	Fired at the Window[p965] before printing
beforematch	Event	Elements	Fired on elements with the hidden=until-found[p862] attribute before they are revealed.
beforetoggle	ToggleEvent[p871]	Elements	Fired on elements with the popover[p925] attribute when they are transitioning between showing and hidden
beforeunload	BeforeUnloadEvent[p1030]	Window[p965]	Fired at the Window[p965] when the page is about to be unloaded, in case the page would like to show a warning prompt
blur	Event	Window[p965], elements	Fired at nodes when they stop being focused[p875]

Event	Interface	Interesting targets	Description
cancel	Event	CloseWatcher[p906], dialog[p675] elements, input[p539] elements	Fired at CloseWatcher[p906] objects or dialog[p675] elements when they receive a close request[p902], or at input[p539] elements whose type[p541] attribute is in the File[p563] state when the user does not change their selection
change	Event	Form controls	Fired at controls when the user commits a value change (see also the input event)
click	PointerEvent	Elements	Normally a mouse event; also synthetically fired at an element before its activation behavior is run, when an element is activated from a non-pointer input device (e.g. a keyboard)
close	Event	CloseWatcher[p906], dialog[p675] elements	Fired at CloseWatcher[p906] objects or dialog[p675] elements when they are closed via a close request[p902] or via web developer code
command	CommandEvent[p872]	Elements	Fired at elements when they handle a user invocation, via a commandfor[p586] attribute.
connect	MessageEvent[p1283]	SharedWorkerGlobalScope[p1324]	Fired at a shared worker's global scope when a new client connects
contextlost	Event	canvas[p713] elements, OffscreenCanvas[p777] objects	Fired when the corresponding CanvasRenderingContext2D[p719] or OffscreenCanvasRenderingContext2D[p781] is lost
contextrestored	Event	canvas[p713] elements, OffscreenCanvas[p777] objects	Fired when the corresponding CanvasRenderingContext2D[p719] or OffscreenCanvasRenderingContext2D[p781] is restored after being lost
currententrychange	NavigationCurrentEntryChangeEvent[p1026]	Navigation[p995]	Fired when navigation.currentEntry[p1002] changes
dispose	Event	NavigationHistoryEntry[p999]	Fired when the session history entry[p1053] corresponding to the NavigationHistoryEntry[p999] has been permanently evicted from session history and can no longer be traversed to
error	Event or ErrorEvent[p1168]	Global scope objects, Worker[p1332] objects, elements, networking-related objects	Fired when unexpected errors occur (e.g. networking errors, script errors, decoding errors)
focus	Event	Window[p965], elements	Fired at nodes gaining focus[p875]
formdata	FormDataEvent[p665]	form[p532] elements	Fired at a form[p532] element when it is constructing the entry list[p661]
hashchange	HashChangeEvent[p1027]	Window[p965]	Fired at the Window[p965] when the fragment part of the document's URL changes
input	Event	Elements	Fired when the user changes the contenteditable[p892] element's content, or the form control's value. See also the change[p1575] event for form controls.
invalid	Event	Form controls	Fired at controls during form validation if they do not satisfy their constraints
languagechange	Event	Global scope objects	Fired at the global scope object when the user's preferred languages change
load	Event	Window[p965], elements	Fired at the Window[p965] when the document has finished loading; fired at an element containing a resource (e.g. img[p359], embed[p413]) when its resource has finished loading
message	MessageEvent[p1283]	Window[p965], EventSource[p1285], MessagePort[p1299], BroadcastChannel[p1303], DedicatedWorkerGlobalScope[p1324], Worker[p1332], ServiceWorkerContainer	Fired at an object when it receives a message
messageerror	MessageEvent[p1283]	Window[p965], MessagePort[p1299], BroadcastChannel[p1303], DedicatedWorkerGlobalScope[p1324], Worker[p1332], ServiceWorkerContainer	Fired at an object when it receives a message that cannot be deserialized
navigate	NavigateEvent[p1013]	Navigation[p995]	Fired before the navigable[p1035] navigates[p1063], reloads[p1077], traverses[p1077], or otherwise[p989] changes its URL
navigateerror	ErrorEvent[p1168]	Navigation[p995]	Fired when a navigation does not complete successfully
navigatesuccess	Event	Navigation[p995]	Fired when a navigation completes successfully

Event	Interface	Interesting targets	Description
offline	Event	Global scope objects	Fired at the global scope object when the network connections fails
online	Event	Global scope objects	Fired at the global scope object when the network connections returns
open	Event	EventSource[p1285]	Fired at EventSource[p1285] objects when a connection is established
pageswap	PageSwapEvent[p1028]	Window[p965]	Fired at the Window[p965] right before a document is unloaded[p1114] as a result of a navigation.
pagehide	PageTransitionEvent[p1029]	Window[p965]	Fired at the Window[p965] when the page's session history entry[p1053] stops being the active entry[p1035]
pagereveal	PageRevealEvent[p1029]	Window[p965]	Fired at the Window[p965] when the page begins to render for the first time after it has been initialized or reactivated[p1101]
pageshow	PageTransitionEvent[p1029]	Window[p965]	Fired at the Window[p965] when the page's session history entry[p1053] becomes the active entry[p1035]
pointercancel	PointerEvent	Elements and Text nodes	Fired at the source node[p919] when the user attempts to initiate a drag-and-drop operation
popstate	PopStateEvent[p1027]	Window[p965]	Fired at the Window[p965] when in some cases of session history traversal[p1077]
readystatechange	Event	Document[p135]	Fired at the Document[p135] when it finishes parsing and again when all its subresources have finished loading
rejectionhandled	PromiseRejectionEvent[p1169]	Global scope objects	Fired at global scope objects when a previously-unhandled promise rejection becomes handled
reset	Event	form[p532] elements	Fired at a form[p532] element when it is reset[p666]
select	Event	Form controls	Fired at form controls when their text selection is adjusted (whether by an API or by the user)
storage	StorageEvent[p1352]	Window[p965]	Fired at Window[p965] event when the corresponding localStorage[p1352] or sessionStorage[p1351] storage areas change
submit	SubmitEvent[p665]	form[p532] elements	Fired at a form[p532] element when it is submitted[p658]
toggle	ToggleEvent[p871]	details[p666] and popover[p925] elements	Fired at details[p666] elements when they open or close; fired on elements with the popover[p925] attribute when they are transitioning between showing and hidden
unhandledrejection	PromiseRejectionEvent[p1169]	Global scope objects	Fired at global scope objects when a promise rejection goes unhandled
unload	Event	Window[p965]	Fired at the Window[p965] object when the page is going away
visibilitychange	Event	Document[p135]	Fired at the Document[p135] object when the page becomes visible or hidden to the user

HTTP headers §[p15] 76

This section is non-normative.

The following HTTP request headers are defined by this specification:

- `Last-Event-ID[p1288]`
- `Ping-From[p326]`
- `Ping-To[p326]`

The following HTTP response headers are defined by this specification:

- `Cross-Origin-Embedder-Policy[p955]`
- `Cross-Origin-Embedder-Policy-Report-Only[p955]`
- `Cross-Origin-Opener-Policy[p946]`
- `Cross-Origin-Opener-Policy-Report-Only[p946]`

- `Origin-Agent-Cluster`
- `Refresh`
- `X-Frame-Options`

MIME types §

This section is non-normative.

The following MIME types are mentioned in this specification:

application/atom+xml
Atom [ATOM]

application/json
JSON [JSON]

application/octet-stream
Generic binary data [RFC2046]

application/microdata+json
Microdata as JSON

application/rss+xml
RSS

application/wasm
WebAssembly [WASM]

application/x-www-form-urlencoded
Form submission

application/xhtml+xml
HTML

application/xml
XML [XML] [RFC7303]

image/gif
GIF images [GIF]

image/jpeg
JPEG images [JPEG]

image/png
PNG images [PNG]

image/svg+xml
SVG images [SVG]

multipart/form-data
Form submission [RFC7578]

multipart/mixed
Generic mixed content [RFC2046]

multipart/x-mixed-replace
Streaming server push

text/css
CSS [CSS]

text/event-stream
Server-sent event streams

text/javascript

JavaScript [JAVASCRIPT]p1583 [RFC9239]p1586

text/json

JSON (legacy type)

text/plain

Generic plain text [RFC2046]p1585 [RFC3676]p1585

text/htmlp1546

HTML

text/pingp1549

Hyperlink auditing

text/uri-list

List of URLs [RFC2483]p1585

text/vcard

vCard [RFC6350]p1586

text/vtt

WebVTT [WEBVTT]p1588

text/xml

XML [XML]p1588 [RFC7303]p1586

video/mp4

MPEG-4 video [RFC4337]p1586

video/mpeg

MPEG video [RFC2046]p1585

References §[p15 79]

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.

[ACCNAME]

Accessible Name and Description Computation, B. Garaventa, M. Sumner. W3C.

[APNG]

(Non-normative) *APNG Specification*. S. Parmenter, V. Vukicevic, A. Smith. Mozilla.

[ARIA]

Accessible Rich Internet Applications (WAI-ARIA), J. Diggs, J. Nurthen, M. Cooper. W3C.

[ARIAHTML]

ARIA in HTML, S. Faulkner, S. O'Hara. W3C.

[ATAG]

(Non-normative) *Authoring Tool Accessibility Guidelines (ATAG) 2.0*, J. Richards, J. Spellman, J. Treviranus. W3C.

[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]

Courbes à poles, P. de Casteljau. INPI, 1959.

[BIDI]

UAX #9: Unicode Bidirectional Algorithm, M. Davis. Unicode Consortium.

[BOCU1]

(Non-normative) *UTN #6: BOCU-1: MIME-Compatible Unicode Compression*, M. Scherer, M. Davis. Unicode Consortium.

[CESU8]

(Non-normative) *UTR #26: Compatibility Encoding Scheme For UTF-16: 8-BIT (CESU-8)*, T. Phipps. Unicode Consortium.

[CHARMOD]

(Non-normative) *Character Model for the World Wide Web 1.0: Fundamentals*, M. Dürst, F. Yergeau, R. Ishida, M. Wolf, T. Texin. W3C.

[CHARMODNORM]

(Non-normative) *Character Model for the World Wide Web: String Matching*, A. Phillips. W3C.

[CLIPBOARD-APIS]

Clipboard API and events, G. Kacmarcik, A. Snigdha. W3C.

[COMPOSITE]

Compositing and Blending, R. Cabanier, N. Andronikos. W3C.

[COMPUTABLE]

(Non-normative) *On computable numbers, with an application to the Entscheidungsproblem*, A. Turing. In *Proceedings of the London Mathematical Society*, series 2, volume 42, pages 230-265. London Mathematical Society, 1937.

[COMPUTEPRESSURE]

(Non-normative) *Compute Pressure*, K. Christiansen, A. Mandy. W3C.

[CONSOLE]

Console, T. Stock, R. Kowalski, D. Farolino. WHATWG.

[COOKIES]

HTTP State Management Mechanism, A. Barth. IETF.

[COOKIESTORE]

(Non-normative) *Cookie Store API*, J. Wilcock, M. Nottingham. WHATWG.

[CREDMAN]

Credential Management, N. Satragno, J. Hodges, M. West. W3C.

[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.

[CSSANCHOR]

CSS Anchor Positioning, T. Atkins, E. Etemad, I. Kilpatrick. W3C.

[CSSANIMATIONS]

CSS Animations, D. Jackson, D. Hyatt, C. Marrin, S. Galineau, L. Baron. W3C.

[CSSATTR]

CSS Style Attributes, T. Çelik, E. Etemad. W3C.

[CSSBG]

CSS Backgrounds and Borders, B. Bos, E. Etemad, B. Kemper. W3C.

[CSSBOX]

CSS Box Model, E. Etemad. W3C.

[CSSCASCADE]

CSS Cascading and Inheritance, E. Etemad, T. Atkins. W3C.

[CSSCONTAIN]

CSS Containment, T. Atkins, F. Rivoal, V. Levin. W3C.

[CSSCOLOR]

CSS Color Module, T. Çelik, C. Lilley, L. Baron. W3C.

[CSSCOLORADJUST]

CSS Color Adjustment Module, E. Etemad, R. Atanassov, R. Lillesveen, T. Atkins. W3C.

[CSSDEVICEADAPT]

CSS Device Adaption, F. Rivoal, M. Rakow. W3C.

[CSSDISPLAY]

CSS Display, T. Atkins, E. Etemad. W3C.

[CSSFONTLOAD]

CSS Font Loading, T. Atkins, J. Daggett. W3C.

[CSSFONTS]

CSS Fonts, J. Daggett. W3C.

[CSSFORMS]

CSS Forms, T. Nguyen. W3C.

[CSSFLEXBOX]

CSS Flexible Box Layout, T. Atkins, E. Etemad, R. Atanassov. W3C.

[CSSGAPS]

CSS Gap Decorations, K. Babbitt. W3C.

[CSSGC]
CSS Generated Content, H. Lie, E. Etemad, I. Hickson. W3C.

[CSSGRID]
CSS Grid Layout, T. Atkins, E. Etemad, R. Atanassov. W3C.

[CSSIMAGES]
CSS Images Module, E. Etemad, T. Atkins, L. Verou. W3C.

[CSSIMAGES4]
CSS Images Module Level 4, E. Etemad, T. Atkins, L. Verou. W3C.

[CSSINLINE]
CSS Inline Layout, D. Cramer, E. Etemad. W3C.

[CSSLISTS]
CSS Lists and Counters, T. Atkins. W3C.

[CSSLOGICAL]
CSS Logical Properties, R. Atanassov, E. Etemad. W3C.

[CSSMULTICOL]
CSS Multi-column Layout, H. Lie, F. Rivoal, R. Andrew. W3C.

[CSSOM]
Cascading Style Sheets Object Model (CSSOM), S. Pieters, G. Adams. W3C.

[CSSOMVIEW]
CSSOM View Module, S. Pieters, G. Adams. W3C.

[CSSOVERFLOW]
CSS Overflow Module, L. Baron, F. Rivoal. W3C.

[CSSPAINT]
(Non-normative) *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 Box Sizing Module, T. Atkins, E. Etemad. W3C.

[CSSSCROLLANCHORING]
(Non-normative) *CSS Scroll Anchoring*, T. Atkins-Bittner. W3C.

[CSSSYNTAX]
CSS Syntax, T. Atkins, S. Sapin. W3C.

[CSSTRANSITIONS]
(Non-normative) *CSS Transitions*, L. Baron, D. Jackson, B. Birtles. 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.

[CSSVIEWTRANSITIONS]

CSS View Transitions, T. Atkins Jr.; J. Archibald; K Sagar. W3C.

[CSSUI]

CSS3 Basic User Interface Module, F. Rivoal. W3C.

[CSSWM]

CSS Writing Modes, E. Etemad, K. Ishii. W3C.

[DASH]

Dynamic adaptive streaming over HTTP (DASH). ISO.

[DEVICEPOSTURE]

(Non-normative) *Device Posture API*, D. Gonzalez-Zuniga, K. Christiansen. W3C.

[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]

Recommendation E.163 — Numbering Plan for The International Telephone Service, CCITT Blue Book, Fascicle II.2, pp. 128-134, November 1988.

[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.

[FETCH-METADATA]

Fetch Metadata Request Headers, M.West. W3C.

[FILEAPI]

File API, A. Ranganathan. W3C.

[FILTERS]

Filter Effects, D. Schulze, D. Jackson, C. Harrelson. W3C.

[FULLSCREEN]

Fullscreen, A. van Kesteren, T. Çelik. WHATWG.

[GEOMETRY]

Geometry Interfaces. S. Pieters, D. Schulze, R. Cabanier. W3C.

[GIF]

(Non-normative) *Graphics Interchange Format*. CompuServe.

[GRAPHICS]

(Non-normative) *Computer Graphics: Principles and Practice in C*, Second Edition, J. Foley, A. van Dam, S. Feiner, J. Hughes. Addison-Wesley. ISBN 0-201-84840-6.

[GREGORIAN]

(Non-normative) *Inter Gravissimas*, A. Lilius, C. Clavius. Gregory XIII Papal Bull, February 1582.

[HRT]

High Resolution Time, I. Grigorik, J. Simonsen, J. Mann. W3C.

[HTMLAAM]

HTML Accessibility API Mappings 1.0, S. Faulkner, A. Surkov, S. O'Hara. W3C.

[HTTP]

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.

[INDEXEDDB]

Indexed Database API, A. Alabbas, J. Bell. W3C.

[INBAND]

Sourcing In-band Media Resource Tracks from Media Containers into HTML, S. Pfeiffer, B. Lund. W3C.

[INFRA]

Infra, A. van Kesteren, D. Denicola. WHATWG.

[INTERSECTIONOBSERVER]

Intersection Observer, S. Zager. W3C.

[RESIZEOBSERVER]

Resize Observer, O. Brufau, E. Álvarez. W3C.

[ISO3166]

ISO 3166: Codes for the representation of names of countries and their subdivisions. ISO.

[ISO4217]

ISO 4217: Codes for the representation of currencies and funds. ISO.

[ISO8601]

(Non-normative) *ISO8601: Data elements and interchange formats — Information interchange — Representation of dates and times*. ISO.

[JAVASCRIPT]

ECMAScript Language Specification. Ecma International.

[JLREQ]

Requirements for Japanese Text Layout. W3C.

[JPEG]

JPEG File Interchange Format, E. Hamilton.

[JSERRORSTACKACCESSOR]

Error Stack Accessor. Ecma International.

[JSERRORSTACKS]

Error Stacks. Ecma International.

[JSDYNAMICCODEBRANDCHECKS]

Dynamic code brand checks. Ecma International.

[JSIMPORTTEXT]

Import Text. Ecma International.

[JSINTL]

ECMAScript Internationalization API Specification. Ecma International.

[JSON]

The JavaScript Object Notation (JSON) Data Interchange Format, T. Bray. IETF.

[JSTEMPORAL]

Temporal. Ecma International.

[LONGTASKS]

Long Tasks, D. Denicola, I. Grigorik, S. Panicker. W3C.

[LONGANIMATIONFRAMES]

Long Animation Frames, N. Rosenthal. W3C.

[MAILTO]

(Non-normative) *The 'mailto' URI scheme*, M. Duerst, L. Masinter, J. Zawinski. IETF.

[MANIFEST]

Web App Manifest, M. Caceres, K. Rohde Christiansen, M. Lamouri, A. Kostiainen, M. Giuca, A. Gustafson. W3C.

[MATHMLCORE]

Mathematical Markup Language (MathML), D. Carlisle, Frédéric Wang. W3C.

[MEDIAFRAG]

Media Fragments URI, R. Troncy, E. Mannens, S. Pfeiffer, D. Van Deursen. W3C.

[MEDIASOURCE]

Media Source Extensions, A. Colwell, A. Bateman, M. Watson. W3C.

[MEDIASTREAM]

Media Capture and Streams, D. Burnett, A. Bergkvist, C. Jennings, A. Narayanan. W3C.

[REPORTING]

Reporting, D. Creager, I. Clelland, M. West. W3C.

[MFREL]

Microformats Wiki: existing rel values. Microformats.

[MIMESNIFF]

MIME Sniffing, G. Hemsley. WHATWG.

[MIX]

Mixed Content, M. West. W3C.

[MNG]

MNG (Multiple-image Network Graphics) Format. G. Randers-Pehrson.

[MPEG2]

ISO/IEC 13818-1: Information technology — Generic coding of moving pictures and associated audio information: Systems. ISO/IEC.

[MPEG4]

ISO/IEC 14496-12: ISO base media file format. ISO/IEC.

[MQ]

Media Queries, H. Lie, T. Çelik, D. Glazman, A. van Kesteren. W3C.

[MULTIPLEBUFFERING]

(Non-normative) *Multiple buffering*. Wikipedia.

[MXSS]

mXSS Attacks: Attacking well-secured Web-Applications by using innerHTML Mutations, M. Heiderich, J. Schwenk, T. Frosch, J. Magazinius, and E. Z. Yang. In *Proceedings of the 2013 ACM SIGSAC Conference on Computer and Communications Security (CCS '13), Berlin, Germany, 2013*.

[NAVIGATIONTIMING]

Navigation Timing, Y. Weiss. W3C.

[NOVARYSEARCH]

The No-Vary-Search HTTP Response Header Field, D. Denicola, J. Roman. IETF.

[NPAPI]

(Non-normative) *Gecko Plugin API Reference*. Mozilla.

[OGGSKELETONHEADERS]

SkeletonHeaders. Xiph.Org.

[OPENSEARCH]

Autodiscovery in HTML/XHTML. In *OpenSearch 1.1 Draft 6*. GitHub.

[ORIGIN]

(Non-normative) *The Web Origin Concept*, A. Barth. IETF.

[PAINTTIMING]

Paint Timing, S. Panicker. W3C.

[PAYMENTREQUEST]

Payment Request API, M. Cáceres, D. Wang, R. Solomakhin, I. Jacobs. W3C.

[PDF]

(Non-normative) *Document management — Portable document format — Part 1: PDF*. ISO.

[PERFORMANCETIMELINE]

Performance Timeline, N. Peña Moreno, W3C.

[PERMISSIONSPOLICY]

Permissions Policy, I. Clelland, W3C.

[PICTUREINPICTURE]

(Non-normative) *Picture-in-Picture*, F. Beaufort, M. Lamouri, W3C

[PINGBACK]

Pingback 1.0, S. Langridge, I. Hickson.

[PNG]

Portable Network Graphics (PNG) Specification, D. Duce. W3C.

[POINTEREVENTS]

Pointer Events, J. Rossi, M. Brubeck, R. Byers, P. H. Lauke. W3C.

[POINTERLOCK]

Pointer Lock, V. Scheib. W3C.

[PPUTF8]

(Non-normative) *The Properties and Promises of UTF-8*, M. Dürst. University of Zürich. In *Proceedings of the 11th International Unicode Conference*.

[PREFETCH]

Prefetch, J. Roman. WICG.

[PRERENDERING-REVAMPED]

(Non-normative) *Prerendering Revamped*, D. Denicola, D. Farolino. W3C.

[PRESENTATION]

Presentation API, M. Foltz, D. Röttsches. W3C.

[REFERRERPOLICY]

Referrer Policy, J. Eisinger, E. Stark. W3C.

[REQUESTIDLECALLBACK]

Cooperative Scheduling of Background Tasks, R. McIlroy, I. Grigorik. W3C.

[RESOURCETIMING]

Resource Timing, Yoav Weiss; Noam Rosenthal. W3C.

[RFC1034]

Domain Names - Concepts and Facilities, P. Mockapetris. IETF, November 1987.

[RFC1123]

Requirements for Internet Hosts -- Application and Support, R. Braden. IETF, October 1989.

[RFC2046]

Multipurpose Internet Mail Extensions (MIME) Part Two: Media Types, N. Freed, N. Borenstein. IETF.

[RFC2397]

The "data" URL scheme, L. Masinter. IETF.

[RFC5545]

Internet Calendaring and Scheduling Core Object Specification (iCalendar), B. Desruisseaux. IETF.

[RFC2483]

URI Resolution Services Necessary for URN Resolution, M. Mealling, R. Daniel. IETF.

[RFC3676]

The Text/Plain Format and DelSp Parameters, R. Gellens. IETF.

[RFC9239]

Updates to ECMAScript Media Types, M. Miller, M. Borins, M. Bynens, B. Farias. IETF.

[RFC4337]

(Non-normative) *MIME Type Registration for MPEG-4*, Y. Lim, D. Singer. IETF.

[RFC7595]

Guidelines and Registration Procedures for URI Schemes, D. Thaler, T. Hansen, T. Hardie. IETF.

[RFC5322]

Internet Message Format, P. Resnick. IETF.

[RFC6381]

The 'Codecs' and 'Profiles' Parameters for "Bucket" Media Types, R. Gellens, D. Singer, P. Frojdh. IETF.

[RFC6266]

Use of the Content-Disposition Header Field in the Hypertext Transfer Protocol (HTTP), J. Reschke. IETF.

[RFC6350]

vCard Format Specification, S. Perreault. IETF.

[RFC6596]

The Canonical Link Relation, M. Ohye, J. Kupke. IETF.

[RFC6903]

Additional Link Relation Types, J. Snell. IETF.

[RFC7034]

(Non-normative) *HTTP Header Field X-Frame-Options*, D. Ross, T. Gondrom. IETF.

[RFC7303]

XML Media Types, H. Thompson, C. Lilley. IETF.

[RFC7578]

Returning Values from Forms: multipart/form-data, L. Masinter. IETF.

[RFC8297]

An HTTP Status Code for Indicating Hints, K. Oku. IETF.

[SCREENORIENTATION]

Screen Orientation, M. Cáceres. W3C.

[SCSU]

(Non-normative) *UTR #6: A Standard Compression Scheme For Unicode*, M. Wolf, K. Whistler, C. Wicksteed, M. Davis, A. Freytag, M. Scherer. Unicode Consortium.

[SECURE-CONTEXTS]

Secure Contexts, M. West. W3C.

[SELECTION]

Selection API, R. Niwa. W3C.

[SELECTORS]

Selectors, E. Etemad, T. Çelik, D. Glazman, I. Hickson, P. Linss, J. Williams. W3C.

[SMS]

(Non-normative) *URI Scheme for Global System for Mobile Communications (GSM) Short Message Service (SMS)*, E. Wilde, A. Vaha-Sipila. IETF.

[STRUCTURED-FIELDS]

Structured Field Values for HTTP, M. Nottingham, P-H. Kamp. IETF.

[SRI]

Subresource Integrity, D. Akhawe, F. Braun, F. Marier, J. Weinberger. W3C.

[STORAGE]

Storage, A. van Kesteren. WHATWG.

[SVG]

Scalable Vector Graphics (SVG) 2, N Andronikos, R. Atanassov, T. Bah, B. Birtles, B. Brinza, C. Concolato, E. Dahlström, C. Lilley, C.

McCormack, D. Schepers, R. Schwerdtfeger, D. Storey, S. Takagi, J. Watt. W3C.

[SW]

Service Workers, A. Russell, J. Song, J. Archibald. W3C.

[TOR]

(Non-normative) *Tor*.

[TOUCH]

Touch Events, D. Schepers, S. Moon, M. Brubeck, A. Barstow, R. Byers. W3C.

[TRUSTED-TYPES]

Trusted Types, K. Kotowicz, M. West. W3C.

[TZDATABASE]

(Non-normative) *Time Zone Database*. IANA.

[UAAG]

(Non-normative) *User Agent Accessibility Guidelines (UAAG) 2.0*, J. Allan, K. Ford, J. Richards, J. Spellman. W3C.

[UIEVENTS]

UI Events Specification, G. Kacmarcik, T. Leithead. W3C.

[UNICODE]

The Unicode Standard. Unicode Consortium.

[UNIVCHARDET]

(Non-normative) *A composite approach to language/encoding detection*, S. Li, K. Momoi. Netscape. In *Proceedings of the 19th International Unicode Conference*.

[URL]

URL, A. van Kesteren. WHATWG.

[URLPATTERN]

URL Pattern, B. Kelly, J. Roman, 宍戸俊哉. WHATWG.

[URN]

URN Syntax, R. Moats. IETF.

[UTF7]

(Non-normative) *UTF-7: A Mail-Safe Transformation Format of Unicode*, D. Goldsmith, M. Davis. IETF.

[UTF8DET]

(Non-normative) *Multilingual form encoding*, M. Dürst. W3C.

[UTR36]

(Non-normative) *UTR #36: Unicode Security Considerations*, M. Davis, M. Suignard. Unicode Consortium.

[WASM]

WebAssembly Core Specification, A. Rossberg. W3C.

[WASMESM]

WebAssembly JavaScript Interface: ESM Integration, L. Clark, D. Ehrenberg., A. Takikawa., G. Bedford. W3C.

[WASMJS]

(Non-normative) *WebAssembly JavaScript Interface*, D. Ehrenberg. W3C.

[WCAG]

(Non-normative) *Web Content Accessibility Guidelines (WCAG)*, A. Kirkpatrick, J. O Connor, A. Campbell, M. Cooper. W3C.

[WEBANIMATIONS]

Web Animations, B. Birtles, S. Stephens, D. Stockwell. W3C.

[WEBAUDIO]

(Non-normative) *Web Audio API*, P. Adenot, H. Choi. W3C.

[WEBAUTHN]

Web Authentication: An API for accessing Public Key Credentials, M. Jones, A. Kumar, E. Lundberg, D. Balfanz, V. Bharadwaj, A. Birgisson, A. Czeskis, J. Hodges, J.C. Jones, H. Le Van Gong, A. Liao, R. Lindemann, J. Bradley, C. Brand, T. Cappalli, A. Langley, G. Mandyam, M. Miller, N. Satragno, N. Steele, J. Tan, S. Weeden, M. West, J. Yasskin. W3C.

[WEBCODECS]

WebCodecs API, C. Cunningham, P. Adenot, B. Aboba. W3C.

[WEBCRYPTO]

Web Cryptography API, D. Huigens. W3C.

[WEBDRIVER]

WebDriver, S. Stewart, D. Burns. W3C.

[WEBDRIVERBIDI]

WebDriver BiDi. W3C

[WEBGL]

WebGL Specifications, D. Jackson, J. Gilbert. Khronos Group.

[WEBGPU]

WebGPU, D. Malyshau, K. Ninomiya. W3C.

[WEBIDL]

Web IDL, E. Chen, T. Gu. WHATWG.

[WEBLINK]

Web Linking, M. Nottingham. IETF.

[WEBLOCKS]

(Non-normative) *Web Locks API*, J. Bell, K. Rosylight. W3C.

[WEBMCG]

WebM Container Guidelines. The WebM Project.

[WEBNFC]

(Non-normative) *Web NFC*, F. Beaufort, K. Christiansen, Z. Kis. W3C.

[WEBRTC]

(Non-normative) *Web RTC*, C. Jennings, F. Castelli, H. Boström, J. Bruaroey. W3C.

[WEBSOCKETS]

WebSockets, A. Rice. WHATWG.

[WEBTRANSPORT]

WebTransport, B. Aboba, N. Jaju, V. Vasiliev. W3C.

[WEBVTT]

WebVTT, S. Pieters. W3C.

[WHATWGWIKI]

The WHATWG Wiki. WHATWG.

[X121]

Recommendation X.121 — International Numbering Plan for Public Data Networks, CCITT Blue Book, Fascicle VIII.3, pp. 317-332.

[XFN]

XFN 1.1 profile, T. Çelik, M. Mullenweg, E. Meyer. GMPG.

[XHR]

`XMLHttpRequest`, A. van Kesteren. WHATWG.

[XKCD1288]

(Non-normative) *Substitutions*, Randall Munroe. xkcd.

[XML]

Extensible Markup Language, T. Bray, J. Paoli, C. Sperberg-McQueen, E. Maler, F. Yergeau. W3C.

[XMLENTITY]

(Non-normative) *XML Entity Definitions for Characters*, D. Carlisle, P. Ion. W3C.

[XMLNS]

Namespaces in XML, T. Bray, D. Hollander, A. Layman, R. Tobin. W3C.

[XMLSSPI]

Associating Style Sheets with XML documents, J. Clark, S. Pieters, H. Thompson. W3C.

[XPATH10]

XML Path Language (XPath) Version 1.0, J. Clark, S. DeRose. W3C.

[XSLT10]

(Non-normative) *XSL Transformations (XSLT) Version 1.0*, J. Clark. W3C.

[XSLTP]

(Non-normative) *DOM XSLTProcessor*, WHATWG Wiki. WHATWG.

Acknowledgments §p15 90

Thanks to Tim Berners-Lee for inventing HTML, without which none of this would exist.

Thanks to Aankhen, Aaqa Ishtyaq, Aaron Boodman, Aaron Leventhal, Aaron Krajeski, Abhishek Ghaskata, Abhishek Gupta, Adam Barth, Alexandra Borovova, Adam de Boor, Adam Hepton, Adam Klein, Adam Rice, Adam Roben, Addison Phillips, Adele Peterson, Adrian Bateman, Adrian Roselli, Adrian Sutton, Agustín Fernández, Aharon (Vladimir) Lanin, Ajai Tirumali, Ajay Poshak, Akash Balenalli, Akatsuki Kitamura, Alan Jeffrey, Alan Plum, Alastair Campbell, Alejandro G. Castro, Alex Bishop, Alex Nicolaou, Alex Nozdriukhin, Alex Rousskov, Alex Soncodi, Alexander Farkas, Alexander J. Vincent, Alexander Kalenik, Alexandre Dieulot, Alexandre Morgaut, Alexey Feldgendler, Алексей Проскуряков (Alexey Proskuryakov), Alexey Shvayka, Alexis Deveria, Alfred Agrell, Ali Juma, Alice Boxhall, Alice Wonder, Allan Clements, Allen Wirfs-Brock, Alex Komoroske, Alex Russell, Alphan Chen, Aman Ansari, Ami Fischman, Amos Jeffries, Amos Lim, Anders Carlsson, André Bargull, André E. Veltstra, Andrea Rendine, Andreas, Andreas Deuschlinger, Andreas Farre, Andreas Kling, Andrei Popescu, Andres Gomez, Andres Rios, Andreu Botella, Andrew Barfield, Andrew Clover, Andrew Gove, Andrew Grieve, Andrew Kaster, Andrew Macpherson, Andrew Oakley, Andrew Paseltiner, Andrew Simons, Andrew Smith, Andrew W. Hagen, Andrew Williams, Andrey V. Lukyanov, Andry Rendy, Andy Davies, Andy Earnshaw, Andy Heydon, Andy Paicu, Andy Palay, Anjana Vakil, Ankur Kaushal, Anna Belle Leiserson, Anna Sidwell, Anthony Boyd, Anthony Bryan, Anthony Hickson, Anthony Ramine, Anthony Ricaud, Anton Vayvod, Antonio Sartori, Antti Koivisto, Arfat Salman, Arkadiusz Michalski, Arne Thomassen, Aron Spohr, Arphen Lin, Arthur Hemery, Arthur Sonzogni, Arthur Stolyar, Arun Patole, Aryeh Gregor, Asanka Herath, Asbjørn Ulsberg, Ashley Gullen, Ashley Sheridan, Asumu Takikawa, Atsushi Takayama, Attila Haraszti, Aurelien Levy, Ave Wrigley, Avi Drissman, Axel Dahmen, 방성범 (Bang Seongbeom), Barry Pollard, Ben Boyle, Ben Godfrey, Ben Golightly, Ben Kelly, Ben Lerner, Ben Leslie, Ben Meadowcroft, Ben Millard, Benjamin Carl Wiley Sittler, Benjamin Hawkes-Lewis, Benjamin VanderSloot, Benji Bilheimer, Benoit Ren, Bert Bos, Bijan Parsia, Bil Corry, Bill Mason, Bill McCoy, Billy Wong, Billy Woods, Bjartur Thorlacius, Björn Höhrmann, Blake Frantz, Bob Lund, Bob Owen, Bobby Holley, Boris Zbarsky, Brad Fults, Brad Neuberg, Brad Spencer, Bradley Meck, Brady Eidson, Brandon Jones, Brendan Eich, Brenton Simpson, Brett Wilson, Brett Zamir, Brian Birtles, Brian Blakely, Brian Campbell, Brian Korver, Brian Kuhn, Brian M. Dube, Brian Ryner, Brian Smith, Brian Wilson, Bryan Sullivan, Bruce Bailey, Bruce D'Arcus, Bruce Lawson, Bruce Miller, Bugs Nash, C. Scott Ananian, C. Williams, Cameron McCormack, Cameron Zemek, Cao Yipeng, Carlos Amengual, Carlos Gabriel Cardona, Carlos Ibarra López, Carlos Perelló Marín, Carolyn MacLeod, Casey Leask, Cătălin Badea, Cătălin Mariș, Cem Turesoy, ceving, Chao Cai, 윤석찬 (Channy Yun), Charl van Niekerk, Charlene Wright, Charles Iliya Krempeaux, Charles McCathie Nevile, Charlie Reis, 白丞祐 (Cheng-You Bai), Chris Apers, Chris Cressman, Chris Dumez, Chris Evans, Chris Harrelson, Chris Markiewicz, Chris Morgan, Chris Morris, Chris Nardi, Chris Needham, Chris Pearce, Chris Peterson, Chris Rebert, Chris Weber, Chris Wilson, Christian Biesinger, Christian Johansen, Christian Schmidt, Christoph Päper, Christophe Dumez, Christopher Aillon, Christopher Cameron, Christopher Ferris, Chriswa, Clark Buehler, Cole Robison, Colin Fine, Collin Jackson, Conrad Crawford, Corey Farwell, Corprew Reed, Craig Cockburn, Csaba Gabor, Csaba Marton, Cynthia Shelly, Cyrille Tuzi, Daksh Shah, Dan Callahan, Dan Yoder, Dane Foster, Daniel Barclay, Daniel Bratell, Daniel Brooks, Daniel Brumbaugh Keeney, Daniel Buchner, Daniel Cheng, Daniel Clark, Daniel Davis, Daniel Ehrenberg, Daniel Ethridge, Daniel Glazman, Daniel Holbert, Daniel Peng, Daniel Schattenkirchner, Daniel Spång, Daniel Steinberg, Daniel Tan, Daniel Trebbien, Daniel Vogelheim, Danny Sullivan, Daphne Preston-Kendal, Darien Maillet Valentine, Darin Adler, Darin Fisher, Darxus, Dave Camp, Dave Cramer, Dave Hodder, Dave Lampton, Dave Singer, Dave Tapuska, Dave Townsend, David Baron, David Bloom, David Bokan, David Bruant, David Carlisle, David E. Cleary, David Egan Evans, David Fink, David Flanagan, David Gerard, David Grogan, David Hale, David Håsäther, David Hyatt, David I. Lehn, David John Burrowes, David Matja, David Remahl, David Resseguie, David Smith, David Storey, David Vest, David Woolley, David Zbarsky, Dave Methvin, DeWitt Clinton, Dean Edridge, Dean Edwards, Dean Jackson, Debanjana Sarkar, Debi Orton, Delan Azabani, Derek Featherstone, Derek Guenther, Devarshi Pant, Devdatta, Devin Mullins, Devin Rousso, Di Zhang, Diego Ferreiro Val, Diego González Zúñiga, Diego Ponce de León, Dimitri Glazkov, Dimitry Golubovsky, Dirk Pranke, Dirk Schulze, Dirkjan Ochtman, Divya Manian, Dmitry Lazutkin, Dmitry Titov, dolphinling, Dominic Cooney, Dominique Hazaël-Massieux, Don Brutzman, Donovan Glover, Doron Rosenberg, Doug Kramer, Doug Simpkinson, Drew Wilson, Edgar Chen, Edmund Lai, Eduard Pascual, Eduardo Vela, Edward Welbourne, Edward Z. Yang, Eemeli Aro, Ehsan Akhgari, Ehsan Karamad, Eira Monstad, Eitan Adler, Eli Friedman, Eli Grey, Eliot Graff, Elisabeth Robson, Elizabeth Castro, Elliott Sprehn, Elliotte Harold, Emilio Cobos Álvarez, Emily Stark, Eric Carlson, Eric Casler, Eric Lawrence, Eric Portis, Eric Rescorla, Eric Semling, Eric Shepherd, Eric Willigers, Erik Arvidsson, Erik Charlebois, Erik Rose, 栗本 英理子 (Eriko Kurimoto), espretto, Evan Jacobs, Evan Martin, Evan Prodromou, Evan Stade, Evert, Evgeny Kapun, ExE-Boss, Ezequiel Garzón, fantasai, Félix Sanz, Felix Sasaki, Fernando Altomare Serboncini, Forbes Lindesay, Francesco Schwarz, Francis Brosnan Blazquez, Franck 'Shift' Quélain, François Marier, Frank Barchard, Frank Liberato, Franklin Shirley, Frederik Braun, Fredrik Söderquist, 鵜飼文敏 (Fumitoshi Ukai), Futomi Hatano, Gavin Carothers, Gavin Kistner, Gareth Rees, Garrett Smith, Gary Blackwood, Gary Kacmarcik, Gary Katsevman, Geoff Richards, Geoffrey Garen, Georg Neis, George Lund, Gianmarco Armellin, Giovanni Campagna, Giuseppe Pascale, Glenn Adams, Glenn Maynard, Graham Klyne, Greg Botten, Greg Houston, Greg Wilkins, Gregg Tavares, Gregory J. Rosmaita, Gregory Terzian, Grey, guest271314, Guilherme Johansson Tramontina, Guy Bedford, Gytis Jakutonis, Håkon Wium Lie, Habib Virji, Hajime Morrita, Hallvord Reiar Michaelsen Steen, Hanna Laakso, Hans S. Tømmerhalt, Hans Stimer, Harald Alvestrand, Hashim Khan, Hayato Ito, 何志翔 (HE Zhixiang), Henri Sivonen, Henrik Lied, Henrik Lievonen, Henry Lewis, Henry Mason, Henry Story, Hermann Donfack Zeufack, 中川博貴 (Hiroki Nakagawa), Hiroshige Hayashizaki, Hiroyuki USHITO, Hitoshi Yoshida, Hongchan Choi, 王华 (Hua Wang), Hugh Bellamy, Hugh Guiney, Hugh Winkler, Ian Bicking, Ian Clelland, Ian Davis, Ian Fette, Ian Henderson, Ian Kilpatrick, Ibrahim Ahmed, Ido Green, Ignacio Javier, Igor Oliveira, 安次嶺 一功 (Ikko Ashimine), Ilya Grigorik, Ingvar Stepanyan, isonmad, Iurii Kucherov, Ivan Enderlin, Ivan Nikulin, Ivan Panchenko, Ivo Emanuel Gonçalves, J. King, J.C. Jones, Jackson Ray Hamilton, Jacob Davies, Jacques Distler, Jake Archibald, Jake

Verbaten, Jakub Vrána, Jakub Łopuszański, Jakub Wilk, James Browning, James Craig, James Graham, James Greene, James Justin Harrell, James Kozianski, James M Snell, James Perrett, James Robinson, Jamie Liu, Jamie Lokier, Jamie Mansfield, Jan Kühle, Jan Miksovsky, Janice Shiu, Janusz Majnert, Jan-Ivar Bruaroey, Jan-Klaas Kollhof, Jared Jacobs, Jason Duell, Jason Kersey, Jason Lustig, Jason Orendorff, Jason White, Jasper Bryant-Greene, Jasper St. Pierre, Jatinder Mann, Jay Henry Kao, Jayson Chen, Jean-Yves Avenard, Jed Hartman, Jeff Balogh, Jeff Cutsinger, Jeff Gilbert, Jeff "=JeffH" Hodges, Jeff Schiller, Jeff Walden, Jeffrey Yasskin, Jeffrey Zeldman, 胡慧鋒 (Jennifer Braithwaite), Jellybean Stonerfish, Jennifer Apacible, Jens Bannmann, Jens Fendler, Jens Oliver Meiert, Jens Widell, Jer Noble, Jeremey Hustman, Jeremy Keith, Jeremy Orlow, Jeremy Roman, Jeroen van der Meer, Jerry Smith, Jesse Renée Beach, Jessica Jong, jfkthame, Jian Li, Jihye Hong, Jim Jewett, Jim Ley, Jim Meehan, Jim Michaels, Jin Dongxun (金东勋, 金東勳, 김동훈), Jinho Bang, Jinjiang (勾三股四), Jirka Kosek, Jjgod Jiang, Joaquim Medeiros, João Eiras, Jochen Eisinger, Joe Clark, Joe Gregorio, Joel Spolsky, Joel Verhagen, Joey Arhar, Johan Herland, Johanna Herman, John Boyer, John Bussjaeger, John Carpenter, John Daggett, John Fallows, John Foliot, John Harding, John Keiser, John Law, John Musgrave, John Snyders, John Stockton, John-Mark Bell, Johnny Stenback, Jon Coppeard, Jon Ferraiolo, Jon Gibbins, Jon Jensen, Jon Perlow, Jonas Sicking, Jonathan Cook, Jonathan Kew, Jonathan Neal, Jonathan Oddy, Jonathan Rees, Jonathan Watt, Jonathan Worent, Jonny Axelsson, Joram Schrijver, Jordan Tucker, Jorgen Horstink, Joris van der Wel, Jorunn Danielsen Newth, Joseph Kesselman, Joseph Mansfield, Joseph Pecoraro, Josh Aas, Josh Hart, Josh Juran, Josh Levenberg, Josh Matthews, Joshua Bell, Joshua Chen, Joshua Randall, Juan Olvera, Juanmi Huertas, Jukka K. Korpela, Jules Clément-Ripoche, Julian Reschke, Julio Lopez, 小勝　純 (Jun Kokatsu), Jun Yang (harttle), Jungkee Song, Jürgen Jeka, Justin Lebar, Justin Novosad, Justin Rogers, Justin Schuh, Justin Sinclair, Juuso Lapinlampi, Ka-Sing Chou, Kagami Sascha Rosylight, Kai Hendry, Kamishetty Sreeja, 呂康豪 (KangHao Lu), Karl Dubost, Karl Tomlinson, Kartik Arora, Kartikaya Gupta, Kathy Walton, 河童エクマ（Kawarabe Ecma） Keith Cirkel, Keith Rollin, Keith Yeung, Kelly Ford, Kelly Norton, Ken Russell, Kenji Baheux, Kevin Babbitt, Kevin Benson, Kevin Cole, Kevin Gadd, Kevin McNee, Kevin Venkiteswaran, Khushal Sagar, Kinuko Yasuda, Koji Ishii, Kornél Pál, Kornel Lesinski, 上野 康平 (UENO, Kouhei), Kris Northfield, Kristian Spangsege, Kristof Zelechovski, Krzysztof Maczyński, 黒澤剛志 (Kurosawa Takeshi), Kurt Catti-Schmidt, Kyle Barnhart, Kyle Hofmann, Kyle Huey, Léonard Bouchet, Léonie Watson, Lachlan Hunt, Larry Masinter, Larry Page, Lars Gunther, Lars Solberg, Laura Carlson, Laura Granka, Laura L. Carlson, Laura Wisewell, Laurens Holst, Lawrence Forooghian, Lee Kowalkowski, Leif Halvard Silli, Leif Kornstaedt, Lenny Domnitser, Leonard Rosenthol, Leons Petrazickis, Liviu Tinta, Lobotom Dysmon, Logan, Logan Moore, Lorenz Ackermann, Loune, Luca Casonato, Lucas Gadani, Łukasz Pilorz, Luke Kenneth Casson Leighton, Luke Warlow, Luke Wilde, Maciej Stachowiak, Magne Andersson, Magnus Kristiansen, Maik Merten, Majid Valipour, Maksim Sadym, Malcolm Rowe, Manish Goregaokar, Manish Tripathi, Manuel Martinez-Almeida, Manuel Rego Casasnovas, Marc Hoyois, Marc-André Choquette, Marc-André Lafortune, Marco Zehe, Marcus Bointon, Marcus Otterström, Marijn Kruisselbrink, Mark Amery, Mark Birbeck, Mark Davis, Mark Green, Mark Miller, Mark Nottingham, Mark Pilgrim, Mark Rogers, Mark Rowe, Mark Schenk, Mark Vickers, Mark Wilton-Jones, Markus Cadonau, Markus Stange, Martijn van der Ven, Martijn Wargers, Martin Atkins, Martin Chaov, Martin Dürst, Martin Honnen, Martin Janecke, Martin Kutschker, Martin Nilsson, Martin Robinson, Martin Thomson, Masataka Yakura, Masatoshi Kimura, Mason Freed, Mason Mize, Mathias Bynens, Mathieu Henri, Matias Larsson, Matt Brubeck, Matt Di Pasquale, Matt Falkenhagen, Matt Giuca, Matt Harding, Matt Schmidt, Matt Wright, Matthew Gaudet, Matthew Gregan, Matthew Mastracci, Matthew Noorenberghe, Matthew Raymond, Matthew Thomas, Matthew Tylee Atkinson, Mattias Waldau, Max Romantschuk, Maxim Tsoy, Mayeul Cantan, Menachem Salomon, Menno van Slooten, Micah Dubinko, Micah Nerren, Michael 'Ratt' Iannarelli, Michael A. Nachbaur, Michael A. Puls II, Michael Carter, Michael Daskalov, Michael Day, Michael Dyck, Michael Enright, Michael Ficarra, Michael Gratton, Michael Kohler, Michael McKelvey, Michael Nordman, Michael Powers, Michael Rakowski, Michael(tm) Smith, Michael Walmsley, Michal Zalewski, Michel Buffa, Michel Fortin, Michelangelo De Simone, Michiel van der Blonk, Miguel Casas-Sanchez, Mihai Şucan, Mihai Parparita, Mike Brown, Mike Dierken, Mike Dixon, Mike Hearn, Mike Pennisi, Mike Schinkel, Mike Shaver, Mikko Rantalainen, Mingye Wang, Mirko Brodesser, Mohamed Zergaoui, Mohammad Al Houssami, Mohammad Reza Zakerinasab, Momdo Nakamura, Morten Stenshorne, Mounir Lamouri, Ms2ger, mtrootyy, 邱慕安 (Mu-An Chiou), Mukilan Thiyagarajan, Mustaq Ahmed, Myles Borins, Nadia Heninger, Nate Chapin, NARUSE Yui, Navid Zolghadr, Neil Deakin, Neil Rashbrook, Neil Soiffer, Nereida Rondon, networkException, Nicholas Shanks, Nicholas Stimpson, Nicholas Zakas, Nickolay Ponomarev, Nicolas Gallagher, Nicolas Pena Moreno, Nicolò Ribaudo, Nidhi Jaju, Nigel Megitt, Nikki Bee, Niklas Gögge, Nina Satragno, Noah Mendelsohn, Noah Slater, Noam Rosenthal, Noel Gordon, Nolan Waite, NoozNooz42, Norbert Lindenberg, Oisín Nolan, Ojan Vafai, Olaf Hoffmann, Olav Junker Kjær, Oldřich Vetešník, Oli Studholme, Oliver Hunt, Oliver Rigby, Olivia (Xiaoni) Lai, Olivier Gendrin, Olli Pettay, Ondřej Žára, Ori Avtalion, Oriol Brufau, oSand, Pablo Flouret, Patrick Dark, Patrick Garies, Patrick H. Lauke, Patrik Persson, Paul Adenot, Paul Lewis, Paul Norman, Per-Erik Brodin, 一丝 (percyley), Perry Smith, Peter Beverloo, Peter Karlsson, Peter Kasting, Peter Moulder, Peter Occil, Peter Stark, Peter Van der Beken, Peter van der Zee, Peter-Paul Koch, Phil Pickering, Philip Ahlberg, Philip Brembeck, Philip Taylor, Philip TAYLOR, Philippe De Ryck, Pierre-Arnaud Allumé, Pierre-Marie Dartus, Pierre-Yves Gérardy, Piers Wombwell, Pooja Sanklecha, Prashant Hiremath, Prashanth Chandra, Prateek Rungta, Pravir Gupta, Prayag Verma, Psychpsyo, 李普君 (Pujun Li), Rachid Finge, Rafael Weinstein, Rafał Miłecki, Rahim Abdi, Rahul Purohit, Raj Doshi, Rajas Moonka, Rakina Zata Amni, Ralf Stoltze, Ralph Giles, Rebecca Star, Remci Mizkur, Remco, Remy Sharp, Rene Saarsoo, Rene Stach, Ric Hardacre, Rich Clark, Rich Doughty, Richa Rupela, Richard Gibson, Richard Ishida, Richard Torres, Ricky Mondello, Rigo Wenning, Rikkert Koppes, Rimantas Liubertas, Riona Macnamara, River Champeimont, Rob Buis, Rob Ennals, Rob Jellinghaus, Rob S, Rob Smith, Robert Blaut, Robert Collins, Robert Hogan, Robert Kieffer, Robert Linder, Robert Millan, Robert O'Callahan, Robert Sayre, Robin Berjon, Robin Schaufler, Rodger Combs, Roland Steiner, Roma Matusevich, Romain Deltour, Roman Ivanov, Roy Fielding, Rune Lillesveen, Russell Bicknell, Ruud Steltenpool, Ryan King, Ryan Landay, Ryan Sleevi, Ryo Kajiwara, Ryo Kato, Ryosuke Niwa, S. Mike Dierken, Salvatore Loreto, Sam Atkins, Sam Dutton, Sam Kuper, Sam Ruby, Sam Sneddon, Sam Weinig, Samikshya Chand, Samuel Bronson, Samy Kamkar, Sander van Lambalgen, Sanjoy Pal, Sanket Joshi, Sarah Gebauer, Sarven Capadisli, Satrujit Behera, Sayan Sivakumaran, Schalk Neethling, Scott Beardsley, Scott González, Scott Hess, Scott Jehl, Scott Miles, Scott O'Hara, Sean B. Palmer, Sean Feng, Sean Fraser, Sean Hayes, Sean Hogan, Sean Knapp, Sebastian Markbåge, Sebastian Schnitzenbaumer, Sendil Kumar N, Seth Call, Seth Dillingham, Shannon Booth, Shannon Moeller, Shanti Rao, Shaun Inman, Shiino Yuki, 贺师俊 (HE Shi-Jun), Shiki Okasaka, Shivani Sharma, shreyateeza, Shubheksha Jalan, Sidak Singh Aulakh, Sierk Bornemann, Sigbjørn Finne, Sigbjørn Vik, Silver Ghost, Silvia Pfeiffer, Šime Vidas, Simon Fraser, Simon Montagu, Simon Sapin, Yu Han, Simon

Spiegel, Simon Wülker, Siye Liu, skeww, Smylers, Srirama Chandra Sekhar Mogali, Stanton McCandlish, stasoid, Stefan Håkansson, Stefan Haustein, Stefan Santesson, Stefan Schumacher, Ştefan Vargyas, Stefan Weiss, Steffen Meschkat, Stephen Chenney, Stephen Ma, Stephen Stewart, Stephen White, Steve Comstock, Steve Faulkner, Steve Fink, Steve Orvell, Steve Runyon, Steven Bennett, Steven Bingler, Steven Garrity, Steven Tate, Stewart Brodie, Stuart Ballard, Stuart Langridge, Stuart Parmenter, Subramanian Peruvemba, Sudhanshu Jaiswal, sudokus999, Sunava Dutta, Surma, Susan Borgrink, Susan Lesch, Sylvain Pasche, T.J. Crowder, Tab Atkins-Bittner, Taiju Tsuiki, Takashi Toyoshima, Takayoshi Kochi, Takeshi Yoshino, Tantek Çelik, 田村健人 (Kent TAMURA), Tawanda Moyo, Taylor Hunt, Ted Mielczarek, Terence Eden, Terrence Wood, Tetsuharu OHZEKI, Theresa O'Connor, Thijs van der Vossen, Thomas Broyer, Thomas Koetter, Thomas O'Connor, Tim Altman, Tim Dresser, Tim Flynn, Tim Johansson, Tim Nguyen, Tim Perry, Tim van der Lippe, TJ VanToll, Tobias Schneider, Tobie Langel, Toby Inkster, Todd Moody, Tom Baker, Tom Pike, Tom Schuster, Tom ten Thij, Tomasz Jakut, Tomek Wytrębowicz, Tommy Thorsen, Tony Ross, Tooru Fujisawa, Toru Kobayashi, Traian Captan, Travis Leithead, Trevor Rowbotham, Trevor Saunders, Trey Eckels, triple-underscore, Tristan Fraipont, Tristan Parisot, 保呂 毅 (Tsuyoshi Horo), Tyler Close, Valentin Gosu, Vardhan Gupta, Vas Sudanagunta, Veli Şenol, Victor Carbune, Victor Costan, Vipul Snehadeep Chawathe, Vitya Muhachev, Vlad Levin, Vladimir Katardjiev, Vladimir Vukićević, Vyacheslav Aristov, voracity, Walter Steiner, Wakaba, Wayne Carr, Wayne Pollock, Wellington Fernando de Macedo, Wenson Hsieh, Weston Ruter, Wilhelm Joys Andersen, Will Levine, Will Ray, William Chen, William Swanson, Willy Martin Aguirre Rodriguez, Wladimir Palant, Wojciech Mach, Wolfram Kriesing, 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, Yoel Hawa, Yonathan Randolph, Yu Huojiang, Yuki Okushi, Yury Delendik, 平野裕 (Yutaka Hirano), Yuzo Fujishima, 西條柚 (Yuzu Saijo), Zhenbin Xu, 张智強 (Zhiqiang Zhang), Zoltan Herczeg, Zyachel, 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`[p713] 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`[p892], 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[p1442] 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`[p355] 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[p785] 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, Alex Russell, Andres Rios, Boris Zbarsky, Brian Kardell, Daniel Buchner, Dominic Cooney, Erik Arvidsson, Elliott Sprehn, Hajime Morrita, 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[p1339]. In particular, thanks to Ian Kilpatrick for his work as editor of the original worklets specification.

Special thanks to the WICG for incubating the speculative loading[p1118] feature. In particular, thanks to Jeremy Roman for his work as editor of the original speculation rules and prefetch specifications.

Special thanks to the WICG for incubating the Sanitizer API[p1233]. In particular, thanks to Frederik Braun, Mario Heiderich, Daniel Vogelheim, and Tom Schuster for their work as editors of the original Sanitizer API specification.

For about ten years starting in 2003, this standard was almost entirely written by Ian Hickson (Google, ian@hixie.ch).

Starting in 2015, the editor group expanded. It is currently maintained by Anne van Kesteren (Apple, annevk@annevk.nl), Domenic

Denicola (Google, d@domenic.me) Dominic Farolino (Google, domfarolino@gmail.com), Philip Jägenstedt (Google, philip@foolip.org), and Simon Pieters (Mozilla, zcorpan@gmail.com).

Intellectual property rights §[p15] 94

The image in the introduction is based on a photo by Wonderlane. (CC BY 2.0)

The image of the wolf in the embedded content introduction is based on a photo by Barry O'Neill. (Public domain)

The image of the kettlebell swing in the embedded content introduction is based on a photo by kokkarina. (CC0 1.0)

The 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`[p687] and `defer`[p687] impact `script`[p685] 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`[p355] 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`[p200] 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[p785] 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`[p175] 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[p1339] feature can be found in the `w3c/css-houdini-drafts` repository, which is available under the W3C Software and Document License.

Part of the revision history of the import maps[p1174] feature can be found in the `WICG/import-maps` repository, which is available under the W3C Software and Document License.

Part of the revision history of the navigation API[p992] feature can be found in the `WICG/navigation-api` repository, which is available under the W3C Software and Document License.

Part of the revision history of the Close requests and close watchers[p902] section can be found in the `WICG/close-watcher` repository, which is available under the W3C Software and Document License.

Part of the revision history of the Speculative loading[p1118] section can be found in the `WICG/nav-speculation` 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. To the extent portions of it are incorporated into source code, such portions in the source code are licensed under the BSD 3-Clause License instead.

This is the Living Standard. Those interested in the patent-review version should view the Living Standard Review Draft.