HTML 5
Draft Recommendation — 9 May 2008

You can take part in this work. Join the working group's discussion list.
Web designers! We have a FAQ, a forum, and a help mailing list for you!

One-page version:

http://www.whatwg.org/specs/web-apps/current-work/

Multiple-page version:

http://www.whatwg.org/specs/web-apps/current-work/multipage/

PDF print versions:

A4: http://www.whatwg.org/specs/web-apps/current-work/html5-a4.pdf
Letter: http://www.whatwg.org/specs/web-apps/current-work/html5-letter.pdf

Version history:

Twitter messages (non-editorial changes only): http://twitter.com/WHATWG
Commit-Watchers mailing list: http://lists.whatwg.org/listinfo.cgi/commit-watchers-whatwg.org
Interactive Web interface: http://html5.org/tools/web-apps-tracker
Subversion interface: http://svn.whatwg.org/
HTML diff with the last version in Subversion: http://whatwg.org/specs/web-apps/current-work/index-diff

Issues:

To send feedback: whatwg@whatwg.org
To view and vote on feedback: http://www.whatwg.org/issues/

Editor:

Ian Hickson, Google, ian@hixie.ch

Abstract

This specification evolves HTML and its related APIs to ease the authoring of Web-based applications. Additions include the context menus, a direct-mode graphics canvas, inline popup windows, and server-sent events. Heavy emphasis is placed on keeping the language backwards compatible with existing legacy user agents and on keeping user agents backwards compatible with existing legacy documents.

Status of this document

This is a work in progress! This document is changing on a daily if not hourly basis in response to comments and as a general part of its development process. Comments are very welcome, please send them to whatwg@whatwg.org. Thank you.

The current focus is in responding to the outstanding feedback. (There is a chart showing current progress.)

Implementors should be aware that this specification is not stable. **Implementors who are not taking part in the discussions are likely to find the specification changing out from under them in incompatible ways.** Vendors interested in implementing this specification before it eventually reaches the call for implementations should join the WHATWG mailing list and take part in the discussions.

This specification is also being produced by the W3C HTML WG. The two specifications are identical from the table of contents onwards.

This specification is intended to replace (be the new version of) what was previously the HTML4, XHTML 1.x, and DOM2 HTML specifications.

Stability

Different parts of this specification are at different levels of maturity.

Some of the more major known issues are marked like this. There are many other issues that have been raised as well; the issues given in this document are not the only known issues! There are also some spec-wide issues that have not yet been addressed: case-sensitivity is a very poorly handled topic right now, and the firing of events needs to be unified (right now some bubble, some don't, they all use different text to fire events, etc). It would also be nice to unify the rules on downloading content when attributes change (e.g. `src` attributes) - should they initiate downloads when the element immediately, is inserted in the document, when active scripts end, etc. This matters e.g. if an attribute is set twice in a row (does it hit the network twice).

Table of contents

1. Introduction
    - 1.1 Scope
        - 1.1.1 Relationship to HTML 4.01, XHTML 1.1, DOM2 HTML
        - 1.1.2 Relationship to XHTML2
        - 1.1.3 Relationship to XUL, Flash, Silverlight, and other proprietary UI languages
    - 1.2 Structure of this specification
        - 1.2.1 How to read this specification
    - 1.3 Conformance requirements
        - 1.3.1 Common conformance requirements for APIs exposed to JavaScript
        - 1.3.2 Dependencies
        - 1.3.3 Features defined in other specifications
    - 1.4 Terminology
        - 1.4.1 HTML vs XHTML

2. The Document Object Model
    - 2.1 Documents
        - 2.1.1 Security
        - 2.1.2 Resource metadata management
    - 2.2 Elements
        - 2.2.1 Reflecting content attributes in DOM attributes
    - 2.3 Common DOM interfaces
        - 2.3.1 Collections
            - 2.3.1.1. HTMLCollection
            - 2.3.1.2. HTMLFormControlsCollection
            - 2.3.1.3. HTMLOptionsCollection
        - 2.3.2 DOMTokenList
        - 2.3.3 DOMStringMap
        - 2.3.4 DOM feature strings
    - 2.4 DOM tree accessors
    - 2.5 Dynamic markup insertion
        - 2.5.1 Controlling the input stream
        - 2.5.2 Dynamic markup insertion in HTML
        - 2.5.3 Dynamic markup insertion in XML
    - 2.6 APIs in HTML documents

3. Semantics and structure of HTML elements
    - 3.1 Introduction
    - 3.2 Common microsyntaxes
        - 3.2.1 Common parser idioms
        - 3.2.2 Boolean attributes
        - 3.2.3 Numbers
            - 3.2.3.1. Unsigned integers
            - 3.2.3.2. Signed integers
            - 3.2.3.3. Real numbers
            - 3.2.3.4. Ratios
            - 3.2.3.5. Percentages and dimensions
            - 3.2.3.6. Lists of integers

- 3.2.4 Dates and times
  - 3.2.4.1. Specific moments in time
  - 3.2.4.2. Vaguer moments in time
- 3.2.5 Time offsets
- 3.2.6 Tokens
- 3.2.7 Keywords and enumerated attributes
- 3.2.8 References
- 3.2.9 URLs

3.3 Documents and document fragments
- 3.3.1 Semantics
- 3.3.2 Structure
- 3.3.3 Kinds of content
  - 3.3.3.1. Metadata content
  - 3.3.3.2. Flow content
  - 3.3.3.3. Sectioning content
  - 3.3.3.4. Heading content
  - 3.3.3.5. Phrasing content
  - 3.3.3.6. Embedded content
  - 3.3.3.7. Interactive content
- 3.3.4 Transparent content models
- 3.3.5 Paragraphs

3.4 Global attributes
- 3.4.1 The `id` attribute
- 3.4.2 The `title` attribute
- 3.4.3 The `lang` (HTML only) and `xml:lang` (XML only) attributes
- 3.4.4 The `dir` attribute
- 3.4.5 The `class` attribute
- 3.4.6 The `irrelevant` attribute
- 3.4.7 The `style` attribute
- 3.4.8 Embedding custom non-visible data

3.5 Interaction
- 3.5.1 Activation
- 3.5.2 Focus
  - 3.5.2.1. Focus management
  - 3.5.2.2. Sequential focus navigation
- 3.5.3 Scrolling elements into view

3.6 The root element
- 3.6.1 The `html` element

3.7 Document metadata
- 3.7.1 The `head` element
- 3.7.2 The `title` element
- 3.7.3 The `base` element
- 3.7.4 The `link` element
- 3.7.5 The `meta` element
  - 3.7.5.1. Standard metadata names
  - 3.7.5.2. Other metadata names
  - 3.7.5.3. Pragma directives
  - 3.7.5.4. Specifying the document's character encoding

- 3.7.6 The `style` element
- 3.7.7 Styling

3.8 Sections
- 3.8.1 The `body` element
- 3.8.2 The `section` element
- 3.8.3 The `nav` element
- 3.8.4 The `article` element
- 3.8.5 The `aside` element
- 3.8.6 The `h1`, `h2`, `h3`, `h4`, `h5`, and `h6` elements
- 3.8.7 The `header` element
- 3.8.8 The `footer` element
- 3.8.9 The `address` element
- 3.8.10 Headings and sections
  - 3.8.10.1. Creating an outline
  - 3.8.10.2. Distinguishing site-wide headings from page headings

3.9 Grouping content
- 3.9.1 The `p` element
- 3.9.2 The `hr` element
- 3.9.3 The `br` element
- 3.9.4 The `pre` element
- 3.9.5 The `dialog` element
- 3.9.6 The `blockquote` element
- 3.9.7 The `ol` element
- 3.9.8 The `ul` element
- 3.9.9 The `li` element
- 3.9.10 The `dl` element
- 3.9.11 The `dt` element
- 3.9.12 The `dd` element

3.10 Text-level semantics
- 3.10.1 The `a` element
- 3.10.2 The `q` element
- 3.10.3 The `cite` element
- 3.10.4 The `em` element
- 3.10.5 The `strong` element
- 3.10.6 The `small` element
- 3.10.7 The `mark` element
- 3.10.8 The `dfn` element
- 3.10.9 The `abbr` element
- 3.10.10 The `time` element
- 3.10.11 The `progress` element
- 3.10.12 The `meter` element
- 3.10.13 The `code` element
- 3.10.14 The `var` element
- 3.10.15 The `samp` element
- 3.10.16 The `kbd` element
- 3.10.17 The `sub` and `sup` elements
- 3.10.18 The `span` element

- 3.10.19 The `i` element
- 3.10.20 The `b` element
- 3.10.21 The `bdo` element
- 3.10.22 Usage summary
- 3.10.23 Footnotes

3.11 Edits
- 3.11.1 The `ins` element
- 3.11.2 The `del` element
- 3.11.3 Attributes common to `ins` and `del` elements
- 3.11.4 Edits and paragraphs
- 3.11.5 Edits and lists

3.12 Embedded content
- 3.12.1 The `figure` element
- 3.12.2 The `img` element
- 3.12.3 The `iframe` element
- 3.12.4 The `embed` element
- 3.12.5 The `object` element
- 3.12.6 The `param` element
- 3.12.7 The `video` element
  - 3.12.7.1. Video and audio codecs for `video` elements
- 3.12.8 The `audio` element
  - 3.12.8.1. Audio codecs for `audio` elements
- 3.12.9 Media elements
  - 3.12.9.1. Error codes
  - 3.12.9.2. Location of the media resource
  - 3.12.9.3. Network states
  - 3.12.9.4. Loading the media resource
  - 3.12.9.5. Offsets into the media resource
  - 3.12.9.6. The ready states
  - 3.12.9.7. Playing the media resource
  - 3.12.9.8. Seeking
  - 3.12.9.9. Cue ranges
  - 3.12.9.10. User interface
  - 3.12.9.11. Time range
  - 3.12.9.12. Event summary
  - 3.12.9.13. Security and privacy considerations
- 3.12.10 The `source` element
- 3.12.11 The `canvas` element
  - 3.12.11.1. The 2D context
    - 3.12.11.1.1. The canvas state
    - 3.12.11.1.2. Transformations
    - 3.12.11.1.3. Compositing
    - 3.12.11.1.4. Colors and styles
    - 3.12.11.1.5. Line styles
    - 3.12.11.1.6. Shadows
    - 3.12.11.1.7. Simple shapes (rectangles)
    - 3.12.11.1.8. Complex shapes (paths)
    - 3.12.11.1.9. Text

3.12.11.1.10. Images
3.12.11.1.11. Pixel manipulation
3.12.11.1.12. Drawing model
3.12.11.2. Color spaces and color correction
3.12.11.3. Security with `canvas` elements
3.12.12 The `map` element
3.12.13 The `area` element
3.12.14 Image maps
3.12.15 MathML
3.12.16 SVG
3.12.17 Dimension attributes
3.13 Tabular data
3.13.1 Introduction
3.13.2 The `table` element
3.13.3 The `caption` element
3.13.4 The `colgroup` element
3.13.5 The `col` element
3.13.6 The `tbody` element
3.13.7 The `thead` element
3.13.8 The `tfoot` element
3.13.9 The `tr` element
3.13.10 The `td` element
3.13.11 The `th` element
3.13.12 Attributes common to `td` and `th` elements
3.13.13 Processing model
3.13.13.1. Forming a table
3.13.13.2. Forming relationships between data cells and header cells
3.14 Forms
3.14.1 The `form` element
3.14.2 The `fieldset` element
3.14.3 The `input` element
3.14.4 The `button` element
3.14.5 The `label` element
3.14.6 The `select` element
3.14.7 The `datalist` element
3.14.8 The `optgroup` element
3.14.9 The `option` element
3.14.10 The `textarea` element
3.14.11 The `output` element
3.14.12 Processing model
3.14.12.1. Form submission
3.15 Scripting
3.15.1 The `script` element
3.15.1.1. Scripting languages
3.15.2 The `noscript` element
3.15.3 The `event-source` element
3.16 Interactive elements

- 3.16.1 The `details` element
- 3.16.2 The `datagrid` element
  - 3.16.2.1. The `datagrid` data model
  - 3.16.2.2. How rows are identified
  - 3.16.2.3. The data provider interface
  - 3.16.2.4. The default data provider
    - 3.16.2.4.1. Common default data provider method definitions for cells
  - 3.16.2.5. Populating the `datagrid` element
  - 3.16.2.6. Updating the `datagrid`
  - 3.16.2.7. Requirements for interactive user agents
  - 3.16.2.8. The selection
  - 3.16.2.9. Columns and captions
- 3.16.3 The `command` element
- 3.16.4 The `menu` element
  - 3.16.4.1. Introduction
  - 3.16.4.2. Building menus and tool bars
  - 3.16.4.3. Context menus
  - 3.16.4.4. Toolbars
- 3.16.5 Commands
  - 3.16.5.1. Using the `a` element to define a command
  - 3.16.5.2. Using the `button` element to define a command
  - 3.16.5.3. Using the `input` element to define a command
  - 3.16.5.4. Using the `option` element to define a command
  - 3.16.5.5. Using the `command` element to define a command

3.17 Data Templates
- 3.17.1 Introduction
- 3.17.2 The `datatemplate` element
- 3.17.3 The `rule` element
- 3.17.4 The `nest` element
- 3.17.5 Global attributes for data templates
- 3.17.6 Processing model
  - 3.17.6.1. The `originalContent` DOM attribute
  - 3.17.6.2. The `template` attribute
  - 3.17.6.3. The `ref` attribute
  - 3.17.6.4. The `NodeDataTemplate` interface
  - 3.17.6.5. Mutations
  - 3.17.6.6. Updating the generated content

3.18 Miscellaneous elements
- 3.18.1 The `legend` element
- 3.18.2 The `div` element

4. Web browsers
- 4.1 Browsing contexts
  - 4.1.1 Nested browsing contexts
  - 4.1.2 Auxiliary browsing contexts
  - 4.1.3 Secondary browsing contexts
  - 4.1.4 Security
  - 4.1.5 Threads

- 4.1.6 Browsing context names
- 4.2 The default view
  - 4.2.1 Security
  - 4.2.2 Constructors
  - 4.2.3 APIs for creating and navigating browsing contexts by name
  - 4.2.4 Accessing other browsing contexts
- 4.3 Origin
  - 4.3.1 Relaxing the same-origin restriction
  - 4.3.2 The string representing the script's domain in IDNA format
- 4.4 Scripting
  - 4.4.1 Running executable code
  - 4.4.2 Security exceptions
  - 4.4.3 The `javascript:` protocol
  - 4.4.4 Events
    - 4.4.4.1. Event handler attributes
    - 4.4.4.2. Event firing
    - 4.4.4.3. Events and the `Window` object
    - 4.4.4.4. Runtime script errors
- 4.5 User prompts
  - 4.5.1 Simple dialogs
  - 4.5.2 Dialogs implemented using separate documents
- 4.6 Browser state
  - 4.6.1 Custom protocol and content handlers
    - 4.6.1.1. Security and privacy
    - 4.6.1.2. Sample user interface
- 4.7 Offline Web applications
  - 4.7.1 Introduction
  - 4.7.2 Application caches
  - 4.7.3 The cache manifest syntax
    - 4.7.3.1. Writing cache manifests
    - 4.7.3.2. Parsing cache manifests
  - 4.7.4 Updating an application cache
  - 4.7.5 Processing model
    - 4.7.5.1. Changes to the networking model
  - 4.7.6 Application cache API
  - 4.7.7 Browser state
- 4.8 Session history and navigation
  - 4.8.1 The session history of browsing contexts
  - 4.8.2 The `History` interface
  - 4.8.3 Activating state object entries
  - 4.8.4 The `Location` interface
    - 4.8.4.1. Security
  - 4.8.5 Implementation notes for session history
- 4.9 Navigating across documents
  - 4.9.1 Page load processing model for HTML files
  - 4.9.2 Page load processing model for XML files
  - 4.9.3 Page load processing model for text files
  - 4.9.4 Page load processing model for images

- 4.9.5 Page load processing model for content that uses plugins
- 4.9.6 Page load processing model for inline content that doesn't have a DOM
- 4.9.7 Navigating to a fragment identifier

4.10 Determining the type of a new resource in a browsing context
- 4.10.1 Content-Type sniffing: text or binary
- 4.10.2 Content-Type sniffing: unknown type
- 4.10.3 Content-Type sniffing: image
- 4.10.4 Content-Type sniffing: feed or HTML
- 4.10.5 Content-Type metadata

4.11 Structured client-side storage
- 4.11.1 Storing name/value pairs
  - 4.11.1.1. Introduction
  - 4.11.1.2. The `Storage` interface
  - 4.11.1.3. The `sessionStorage` attribute
  - 4.11.1.4. The `localStorage` attribute
  - 4.11.1.5. The `storage` event
    - 4.11.1.5.1. Event definition
  - 4.11.1.6. Threads
- 4.11.2 Database storage
  - 4.11.2.1. Introduction
  - 4.11.2.2. Databases
  - 4.11.2.3. Executing SQL statements
  - 4.11.2.4. Database query results
  - 4.11.2.5. Errors
  - 4.11.2.6. Processing model
- 4.11.3 Disk space
- 4.11.4 Privacy
  - 4.11.4.1. User tracking
  - 4.11.4.2. Cookie resurrection
- 4.11.5 Security
  - 4.11.5.1. DNS spoofing attacks
  - 4.11.5.2. Cross-directory attacks
  - 4.11.5.3. Implementation risks
  - 4.11.5.4. SQL and user agents
  - 4.11.5.5. SQL injection

4.12 Links
- 4.12.1 Hyperlink elements
- 4.12.2 Following hyperlinks
  - 4.12.2.1. Hyperlink auditing
- 4.12.3 Link types
  - 4.12.3.1. Link type "`alternate`"
  - 4.12.3.2. Link type "`archives`"
  - 4.12.3.3. Link type "`author`"
  - 4.12.3.4. Link type "`bookmark`"
  - 4.12.3.5. Link type "`contact`"
  - 4.12.3.6. Link type "`external`"
  - 4.12.3.7. Link type "`feed`"
  - 4.12.3.8. Link type "`help`"

- 4.12.3.9. Link type "`icon`"
- 4.12.3.10. Link type "`license`"
- 4.12.3.11. Link type "`nofollow`"
- 4.12.3.12. Link type "`noreferrer`"
- 4.12.3.13. Link type "`pingback`"
- 4.12.3.14. Link type "`prefetch`"
- 4.12.3.15. Link type "`search`"
- 4.12.3.16. Link type "`stylesheet`"
- 4.12.3.17. Link type "`sidebar`"
- 4.12.3.18. Link type "`tag`"
- 4.12.3.19. Hierarchical link types
  - 4.12.3.19.1. Link type "`index`"
  - 4.12.3.19.2. Link type "`up`"
- 4.12.3.20. Sequential link types
  - 4.12.3.20.1. Link type "`first`"
  - 4.12.3.20.2. Link type "`last`"
  - 4.12.3.20.3. Link type "`next`"
  - 4.12.3.20.4. Link type "`prev`"
- 4.12.3.21. Other link types

4.13 Interfaces for URI manipulation

5. Editing
- 5.1 Introduction
- 5.2 The `contenteditable` attribute
  - 5.2.1 User editing actions
  - 5.2.2 Making entire documents editable
- 5.3 Drag and drop
  - 5.3.1 Introduction
  - 5.3.2 The `DragEvent` and `DataTransfer` interfaces
  - 5.3.3 Events fired during a drag-and-drop action
  - 5.3.4 Drag-and-drop processing model
    - 5.3.4.1. When the drag-and-drop operation starts or ends in another document
    - 5.3.4.2. When the drag-and-drop operation starts or ends in another application
  - 5.3.5 The `draggable` attribute
  - 5.3.6 Copy and paste
    - 5.3.6.1. Copy to clipboard
    - 5.3.6.2. Cut to clipboard
    - 5.3.6.3. Paste from clipboard
    - 5.3.6.4. Paste from selection
  - 5.3.7 Security risks in the drag-and-drop model
- 5.4 Undo history
  - 5.4.1 The `UndoManager` interface
  - 5.4.2 Undo: moving back in the undo transaction history
  - 5.4.3 Redo: moving forward in the undo transaction history
  - 5.4.4 The `UndoManagerEvent` interface and the `undo` and `redo` events
  - 5.4.5 Implementation notes
- 5.5 The text selection APIs
  - 5.5.1 APIs for the browsing context selection

        - 5.5.2 APIs for the text field selections
    - 5.6 Command APIs
- 6. Communication
    - 6.1 Event definitions
    - 6.2 Server-sent DOM events
        - 6.2.1 The `RemoteEventTarget` interface
        - 6.2.2 Connecting to an event stream
        - 6.2.3 Parsing an event stream
        - 6.2.4 Interpreting an event stream
        - 6.2.5 Notes
    - 6.3 Network connections
        - 6.3.1 Introduction
        - 6.3.2 The `Connection` interface
        - 6.3.3 Connection Events
        - 6.3.4 TCP connections
        - 6.3.5 Broadcast connections
            - 6.3.5.1. Broadcasting over TCP/IP
            - 6.3.5.2. Broadcasting over Bluetooth
            - 6.3.5.3. Broadcasting over IrDA
        - 6.3.6 Peer-to-peer connections
            - 6.3.6.1. Peer-to-peer connections over TCP/IP
            - 6.3.6.2. Peer-to-peer connections over Bluetooth
            - 6.3.6.3. Peer-to-peer connections over IrDA
        - 6.3.7 The common protocol for TCP-based connections
            - 6.3.7.1. Clients connecting over TCP
            - 6.3.7.2. Servers accepting connections over TCP
            - 6.3.7.3. Sending and receiving data over TCP
        - 6.3.8 Security
        - 6.3.9 Relationship to other standards
    - 6.4 Cross-document messaging
        - 6.4.1 Processing model
- 7. Repetition templates
- 8. The HTML syntax
    - 8.1 Writing HTML documents
        - 8.1.1 The DOCTYPE
        - 8.1.2 Elements
            - 8.1.2.1. Start tags
            - 8.1.2.2. End tags
            - 8.1.2.3. Attributes
            - 8.1.2.4. Optional tags
            - 8.1.2.5. Restrictions on content models
            - 8.1.2.6. Restrictions on the contents of CDATA and RCDATA elements
        - 8.1.3 Text
            - 8.1.3.1. Newlines
        - 8.1.4 Character references
        - 8.1.5 CDATA blocks
        - 8.1.6 Comments

- 8.2 Parsing HTML documents
  - 8.2.1 Overview of the parsing model
  - 8.2.2 The input stream
    - 8.2.2.1. Determining the character encoding
    - 8.2.2.2. Character encoding requirements
    - 8.2.2.3. Preprocessing the input stream
    - 8.2.2.4. Changing the encoding while parsing
  - 8.2.3 Parse state
    - 8.2.3.1. The insertion mode
    - 8.2.3.2. The stack of open elements
    - 8.2.3.3. The list of active formatting elements
    - 8.2.3.4. The element pointers
  - 8.2.4 Tokenisation
    - 8.2.4.1. Tokenising character references
  - 8.2.5 Tree construction
    - 8.2.5.1. Creating and inserting elements
    - 8.2.5.2. Closing elements that have implied end tags
    - 8.2.5.3. The "initial" insertion mode
    - 8.2.5.4. The "before html" insertion mode
    - 8.2.5.5. The "before head" insertion mode
    - 8.2.5.6. The "in head" insertion mode
    - 8.2.5.7. The "in head noscript" insertion mode
    - 8.2.5.8. The "after head" insertion mode
    - 8.2.5.9. The "in body" insertion mode
    - 8.2.5.10. The "in table" insertion mode
    - 8.2.5.11. The "in caption" insertion mode
    - 8.2.5.12. The "in column group" insertion mode
    - 8.2.5.13. The "in table body" insertion mode
    - 8.2.5.14. The "in row" insertion mode
    - 8.2.5.15. The "in cell" insertion mode
    - 8.2.5.16. The "in select" insertion mode
    - 8.2.5.17. The "in select in table" insertion mode
    - 8.2.5.18. The "in foreign content" insertion mode
    - 8.2.5.19. The "after body" insertion mode
    - 8.2.5.20. The "in frameset" insertion mode
    - 8.2.5.21. The "after frameset" insertion mode
    - 8.2.5.22. The "after after body" insertion mode
    - 8.2.5.23. The "after after frameset" insertion mode
  - 8.2.6 The end
- 8.3 Namespaces
- 8.4 Serialising HTML fragments
- 8.5 Parsing HTML fragments
- 8.6 Named character references

9. Rendering and user-agent behavior
- 9.1 Rendering and the DOM
- 9.2 Rendering and menus/toolbars
  - 9.2.1 The 'icon' property
- 9.3 Obsolete elements, attributes, and APIs
  - 9.3.1 The `body` element

9.3.2 The `applet` element

10. Things that you can't do with this specification because they are better handled using other technologies that are further described herein
10.1 Localisation
10.2 Declarative 2D vector graphics and animation
10.3 Declarative 3D scenes
10.4 Timers

Index

References

Acknowledgements

1. Introduction

This section is non-normative.

The World Wide Web's markup language has always been HTML. HTML was primarily designed as a language for semantically describing scientific documents, although its general design and adaptations over the years has enabled it to be used to describe a number of other types of documents.

The main area that has not been adequately addressed by HTML is a vague subject referred to as Web Applications. This specification attempts to rectify this, while at the same time updating the HTML specifications to address issues raised in the past few years.

1.1 Scope

This section is non-normative.

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

The scope of this specification does not include addressing presentation concerns (although default rendering rules for Web browsers are included at the end of this specification).

The scope of this specification does not include documenting every HTML or DOM feature supported by Web browsers. Browsers support many features that are considered to be very bad for accessibility or that are otherwise inappropriate. For example, the `blink` element is clearly presentational and authors wishing to cause text to blink should instead use CSS.

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. For instance online purchasing systems, searching systems, games (especially multiplayer online games), public telephone books or address books, communications software (e-mail clients, instant messaging clients, discussion software), document editing software, etc.

For sophisticated cross-platform applications, there already exist several proprietary solutions (such as Mozilla's XUL, Adobe's Flash, or Microsoft's Silverlight). These solutions are evolving faster than any standards process could follow, and the requirements are evolving even faster. These systems are also significantly more complicated to specify, and are orders of magnitude more difficult to achieve interoperability with, than the solutions described in this document. Platform-specific solutions for such sophisticated applications (for example the MacOS X Core APIs) are even further ahead.

1.1.1 Relationship to HTML 4.01, XHTML 1.1, DOM2 HTML

This section is non-normative.

This specification represents a new version of HTML4 and XHTML1, along with a new version of the associated DOM2 HTML API. Migration from HTML4 or XHTML1 to the format and APIs described in this specification should in most cases be straightforward, as care has been taken to ensure that backwards-compatibility is retained.

This specification will eventually supplant Web Forms 2.0 as well. [WF2]

1.1.2 Relationship to XHTML2

This section is non-normative.

XHTML2 [XHTML2] defines a new HTML vocabulary with better features for hyperlinks, multimedia content, annotating document edits, rich metadata, declarative interactive forms, and describing the semantics of human literary works such as poems and scientific papers.

However, it lacks elements to express the semantics of many of the non-document types of content often seen on the Web. For instance, forum sites, auction sites, search engines, online shops, and the like, do not fit the document metaphor well, and are not covered by XHTML2.

This specification aims to extend HTML so that it is also suitable in these contexts.

XHTML2 and this specification use different namespaces and therefore can both be implemented in the same XML processor.

1.1.3 Relationship to XUL, Flash, Silverlight, and other proprietary UI languages

This section is non-normative.

This specification is independent of the various proprietary UI languages that various vendors provide. As an open, vendor-neutral language, HTML provides for a solution to the same problems without the risk of vendor lock-in.

1.2 Structure of this specification

This section is non-normative.

This specification is divided into the following important sections:

The DOM
: The DOM, or Document Object Model, provides a base for the rest of the specification.

The Semantics
: Documents are built from elements. These elements form a tree using the DOM. Each element also has a predefined meaning, which is explained in this section. User agent requirements for how to handle each element are also given, along with rules for authors on how to use the element.

Browsing Contexts
: HTML documents do not exist in a vacuum — this section defines many of the features that affect environments that deal with multiple pages, links between pages, and running scripts.

APIs
: The Editing APIs: HTML documents can provide a number of mechanisms for users to modify content, which are described in this section.
: The Communication APIs: Applications written in HTML often require mechanisms to communicate with remote servers, as well as communicating with other applications from different domains running on the

same client.
Repetition Templates: A mechanism to support repeating sections in forms.

The Language Syntax

All of these features would be for naught if they couldn't be represented in a serialised form and sent to other people, and so this section defines the syntax of HTML, along with rules for how to parse HTML.

There are also a couple of appendices, defining rendering rules for Web browsers and listing areas that are out of scope for this specification.

1.2.1 How to read this specification

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

1.3 Conformance requirements

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

The key words "MUST", "MUST NOT", "REQUIRED", "SHOULD", "SHOULD NOT", "RECOMMENDED", "MAY", and "OPTIONAL" in the normative parts of this document are to be interpreted as described in RFC2119. For readability, these words do not appear in all uppercase letters in this specification. [RFC2119]

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

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

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 XHTML 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.

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

Web browsers that support HTML must process documents labelled as `text/html` as described in this specification, so that users can interact with them.

Non-interactive presentation user agents

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

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

> 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.

User agents with no scripting support

Implementations that do not support scripting (or which have their scripting features disabled) 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. Conformance checkers are exempt from detecting errors that require interpretation of the author's intent (for example, while a document is non-conforming if the content of a `blockquote` element is not a quote, conformance checkers do not have to check that `blockquote` elements only contain quoted material).

Conformance checkers must check that the input document conforms when scripting is disabled, and should also check that the input document conforms when scripting is enabled. (This is only a "SHOULD" and not a "MUST" requirement because it has been proven to be impossible. [HALTINGPROBLEM])

The term "HTML5 validator" can be used to refer to a conformance checker that itself conforms to the applicable requirements of this specification.

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

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

1. ***Criteria that can be expressed in a DTD.***

2. ***Criteria that cannot be expressed by a DTD, but can still be checked by a***

machine.

3. *Criteria that can only be checked by a human.*

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

Data mining tools

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

> A tool that generates document outlines but increases the nesting level for each paragraph and does not increase the nesting level for each section would not be conforming.

Authoring tools and markup generators

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

Authoring tools are exempt from the strict requirements of using elements only for their specified purpose, but only to the extent that authoring tools are not yet able to determine author intent.

> For example, it is not conforming to use an `address` element for arbitrary contact information; that element can only be used for marking up contact information for the author of the document or section. However, since an authoring tools is likely unable to determine the difference, an authoring tool is exempt from that requirement.

Note: In terms of conformance checking, an editor is therefore required to output documents that conform to the same extent that a conformance checker will verify.

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

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

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

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

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

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. The former category of requirements are requirements on documents and authoring tools. The second category are requirements on user agents.

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

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

For compatibility with existing content and prior specifications, this specification describes two authoring formats: one based on XML (referred to as **XHTML5**), and one using a custom format inspired by SGML (referred to as **HTML5**). Implementations may support only one of these two formats, although supporting both is encouraged.

XHTML documents (XML documents using elements from the HTML namespace) that use the new features described in this specification and that are served over the wire (e.g. by HTTP) must be sent using an XML MIME type such as `application/xml` or `application/xhtml+xml` and must not be served as `text/html`. [RFC3023]

Such XML documents may contain a `DOCTYPE` if desired, but this is not required to conform to this specification.

Note: According to the XML specification, XML processors are not guaranteed to process the external DTD subset referenced in the DOCTYPE. This means, for example, that using entities for characters in XHTML documents is unsafe (except for &lt;, &gt;, &amp;, &quot; and &apos;). For interoperability, authors are advised to avoid optional features of XML.

HTML documents, if they are served over the wire (e.g. by HTTP) must be labelled with the `text/html` MIME type.

The language in this specification assumes that the user agent expands all entity references, and therefore does not include entity reference nodes in the DOM. If user agents do include entity reference nodes in the DOM, then user agents must handle them as if they were fully expanded when implementing this specification. For example, if a requirement talks about an element's child text nodes, then any text nodes that are children of an entity reference that is a child of that element would be used as well. Entity references to unknown entities must be treated as if they contained just an empty text node for the purposes of the algorithms defined in this specification.

1.3.1 Common conformance requirements for APIs exposed to JavaScript

A lot of arrays/lists/collections in this spec assume zero-based indexes but use the term "*index*th" liberally. We should define those to be zero-based and be clearer about this.

Unless otherwise specified, if a DOM attribute that is a floating point number type (`float`) is assigned an Infinity or Not-a-Number value, a `NOT_SUPPORTED_ERR` exception must be raised.

Unless otherwise specified, if a method with an argument that is a floating point number type (`float`) is passed an Infinity or Not-a-Number value, a `NOT_SUPPORTED_ERR` exception must be raised.

Unless otherwise specified, if a method is passed fewer arguments than is defined for that method in its IDL definition, a `NOT_SUPPORTED_ERR` exception must be raised.

Unless otherwise specified, if a method is passed more arguments than is defined for that method in its IDL definition, the excess arguments must be ignored.

1.3.2 Dependencies

This specification relies on several other underlying specifications.

XML

Implementations that support XHTML5 must support some version of XML, as well as its corresponding namespaces specification, because XHTML5 uses an XML serialisation with namespaces. [XML] [XMLNAMES]

XML Base

User agents must follow the rules given by XML Base to resolve relative URIs in HTML and XHTML fragments. That is the mechanism used in this specification for resolving relative URIs in DOM trees. [XMLBASE]

Note: It is possible for `xml:base` attributes to be present even in HTML fragments, as such attributes can be added dynamically using script.

DOM

Implementations must support some version of DOM Core and DOM Events, because this specification is defined in terms of the DOM, and some of the features are defined as extensions to the DOM Core interfaces. [DOM3CORE] [DOM3EVENTS]

ECMAScript

Implementations that use ECMAScript to implement the APIs defined in this specification must implement them in a manner consistent with the ECMAScript Bindings defined in the Web IDL specification, as this specification uses that specification's terminology. [WebIDL]

Media Queries

Implementations must support some version of the Media Queries language. However, when applying the rules of the Media Queries specification to media queries found in content attributes of HTML elements, user agents must act as if all U+000B LINE TABULATION characters in the attribute were in fact U+0020 SPACE characters. This is required to provide a consistent processing of space characters in HTML. [MQ]

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

Note: This specification might have certain additional requirements on character encodings,

image formats, audio formats, and video formats in the respective sections.

1.3.3 Features defined in other specifications

Some elements are defined in terms of their DOM **`textContent`** attribute. This is an attribute defined on the `Node` interface in DOM3 Core. [DOM3CORE]

Should textContent be defined differently for dir="" and <bdo>? Should we come up with an alternative to textContent that handles those and other things, like alt=""?

The interface **`DOMTimeStamp`** is defined in DOM3 Core. [DOM3CORE]

The term **activation behavior** is used as defined in the DOM3 Events specification. [DOM3EVENTS] At the time of writing, DOM3 Events hadn't yet been updated to define that phrase.

The rules for handling alternative style sheets are defined in the CSS object model specification. [CSSOM]

See http://dev.w3.org/cvsweb/~checkout~/csswg/cssom/Overview.html?rev=1.35&content-type=text/html;%20charset=utf-8

Certain features are defined in terms of CSS <color> values. When the CSS value `currentColor` is specified in this context, the "computed value of the 'color' property" for the purposes of determining the computed value of the `currentColor` keyword is the computed value of the 'color' property on the element in question. [CSS3COLOR]

> If a canvas gradient's `addColorStop()` method is called with the `currentColor` keyword as the color, then the computed value of the 'color' property on the `canvas` element is the one that is used.

1.4 Terminology

This specification refers to both HTML and XML attributes and DOM 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 **DOM attributes** for those from the DOM. Similarly, the term "properties" is used for both ECMAScript object properties and CSS properties. When these are ambiguous they are qualified as object properties and CSS properties respectively.

To ease migration from HTML to XHTML, UAs conforming to this specification will place elements in HTML in the `http://www.w3.org/1999/xhtml` namespace, at least for the purposes of the DOM and CSS. The term "**elements in the HTML namespace**", or "**HTML elements**" for short, when used in this specification, thus refers to both HTML and XHTML elements.

Unless otherwise stated, all elements defined or mentioned in this specification are in the `http://www.w3.org/1999/xhtml` namespace, and all attributes defined or mentioned in this specification have no namespace (they are in the per-element partition).

The term HTML documents is sometimes used in contrast with XML documents to specifically mean documents that were parsed using an HTML parser (as opposed to using an XML parser or created purely through the DOM).

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

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

For readability, the term URI is used to refer to both ASCII URIs and Unicode IRIs, as those terms are defined by RFC 3986 and RFC 3987 respectively. On the rare occasions where IRIs are not allowed but ASCII URIs are, this is called out explicitly. [RFC3986] [RFC3987]

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

An element is said to have been **inserted into a document** when its root element changes and is now the document's root element.

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

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

When an XML name, such as an attribute or element name, is referred to in the form *`prefix`*`:`*`localName`*, as in `xml:id` or `svg:rect`, it refers to a name with the local name *localName* and the namespace given by the prefix, as defined by the following table:

`xml`

    `http://www.w3.org/XML/1998/namespace`

`html`

    `http://www.w3.org/1999/xhtml`

`svg`

    `http://www.w3.org/2000/svg`

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.

DOM interfaces defined in this specification use Web IDL. User agents must implement these interfaces as defined by the Web IDL specification. [WEBIDL]

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`".

A DOM 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 that means that any attributes returning that object must always return the same object (not a new object each time), and the attributes and methods on that object must operate on the actual underlying data, not a snapshot of the data.

The terms *fire* and *dispatch* are used interchangeably in the context of events, as in the DOM Events specifications. [DOM3EVENTS]

The term **text node** refers to any `Text` node, including `CDATASection` nodes; specifically, any `Node` with node type `TEXT_NODE` (3) or `CDATA_SECTION_NODE` (4). [DOM3CORE]

Some of the algorithms in this specification, for historical reasons, require the user agent to **pause** until some condition has been met. While a user agent is paused, it must ensure that no scripts execute (e.g. no event handlers, no timers, etc). User agents should remain responsive to user input while paused, however, albeit without letting the user interact with Web pages where that would involve invoking any script.

1.4.1 HTML vs XHTML

This section is non-normative.

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

The in-memory representation is known as "DOM5 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 "HTML5". This is the format recommended for most authors. It is compatible with all legacy Web browsers. If a document is transmitted with the MIME type `text/html`, then it will be processed as an "HTML5" document by Web browsers.

The second concrete syntax uses XML, and is known as "XHTML5". When a document is transmitted with an XML MIME type, such as `application/xhtml+xml`, then it is processed by an XML processor by Web browsers, and treated as an "XHTML5" document. Authors are reminded that the processing for XML and HTML differs; in particular, even minor syntax errors will prevent an XML document from being rendered fully, whereas they would be ignored in the "HTML5" syntax.

The "DOM5 HTML", "HTML5", and "XHTML5" representations cannot all represent the same content. For example, namespaces cannot be represented using "HTML5", but they are supported in "DOM5 HTML" and "XHTML5". Similarly, documents that use the `noscript` feature can be represented using "HTML5", but cannot be represented with "XHTML5" and "DOM5 HTML". Comments that contain the string "`-->`" can be represented in "DOM5 HTML" but not in "HTML5" and "XHTML5". And so forth.

2. The Document Object Model

The Document Object Model (DOM) is a representation — a model — of a document and its content. [DOM3CORE] 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.

This specification defines the language represented in the DOM by features together called DOM5 HTML. DOM5 HTML consists of DOM Core `Document` nodes and DOM Core `Element` nodes, along with text nodes and other content.

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

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

In addition, documents and elements in the DOM host APIs that extend the DOM Core APIs, providing new features to application developers using DOM5 HTML.

2.1 Documents

Every XML and HTML document in an HTML UA is represented by a `Document` object. [DOM3CORE]

`Document` objects are assumed to be **XML documents** unless they are flagged as being **HTML documents** when they are created. Whether a document is an HTML document or an XML document affects the behavior of certain APIs, as well as a few CSS rendering rules. [CSS21]

Note: A `Document` object created by the `createDocument()` API on the `DOMImplementation` object is initially an XML document, but can be made into an HTML document by calling `document.open()` on it.

All `Document` objects (in user agents implementing this specification) must also implement the `HTMLDocument` interface, available using binding-specific methods. (This is the case whether or not the document in question is an HTML document or indeed whether it contains any HTML elements at all.) `Document` objects must also implement the document-level interface of any other namespaces found in the document that the UA supports. For example, if an HTML implementation also supports SVG, then the `Document` object must implement `HTMLDocument` and `SVGDocument`.

Note: Because the `HTMLDocument` interface is now obtained using binding-specific casting methods instead of simply being the primary interface of the document object, it is no longer defined as inheriting from `Document`.

```
interface HTMLDocument {
  // Resource metadata management
  [PutForwards=href] readonly attribute Location location;
  readonly attribute DOMString URL;
           attribute DOMString domain;
  readonly attribute DOMString referrer;
           attribute DOMString cookie;
  readonly attribute DOMString lastModified;
  readonly attribute DOMString compatMode;
           attribute DOMString charset;
  readonly attribute DOMString characterSet;
  readonly attribute DOMString defaultCharset;
  readonly attribute DOMString readyState;

  // DOM tree accessors
           attribute DOMString title;
           attribute DOMString dir;
           attribute HTMLElement body;
  readonly attribute HTMLCollection images;
  readonly attribute HTMLCollection embeds;
  readonly attribute HTMLCollection plugins;
  readonly attribute HTMLCollection links;
  readonly attribute HTMLCollection forms;
  readonly attribute HTMLCollection anchors;
  readonly attribute HTMLCollection scripts;
  NodeList getElementsByName(in DOMString elementName);
  NodeList getElementsByClassName(in DOMString classNames);

  // Dynamic markup insertion
           attribute DOMString innerHTML;
  HTMLDocument open();
  HTMLDocument open(in DOMString type);
  HTMLDocument open(in DOMString type, in DOMString replace);
  Window open(in DOMString url, in DOMString name, in DOMString features);
  Window open(in DOMString url, in DOMString name, in DOMString features, in
boolean replace);
  void close();
  void write(in DOMString text);
  void writeln(in DOMString text);

  // Interaction
  readonly attribute Element activeElement;
  readonly attribute boolean hasFocus;

  // Commands
  readonly attribute HTMLCollection commands;

  // Editing
           attribute boolean designMode;
  boolean execCommand(in DOMString commandId);
  boolean execCommand(in DOMString commandId, in boolean showUI);
  boolean execCommand(in DOMString commandId, in boolean showUI, in
DOMString value);
  boolean queryCommandEnabled(in DOMString commandId);
  boolean queryCommandIndeterm(in DOMString commandId);
  boolean queryCommandState(in DOMString commandId);
  boolean queryCommandSupported(in DOMString commandId);
  DOMString queryCommandValue(in DOMString commandId);
  Selection getSelection();
};
```

Since the `HTMLDocument` interface holds methods and attributes related to a number of disparate features, the members of this interface are described in various different sections.

2.1.1 Security

User agents must raise a security exception whenever any of the members of an `HTMLDocument` object are accessed by scripts whose effective script origin is not the same as the `Document`'s effective script origin.

2.1.2 Resource metadata management

The **`URL`** attribute must return the document's address.

The **`referrer`** attribute must return either the URI of the page which navigated the browsing context to the current document (if any), or the empty string if there is no such originating page, or if the UA has been configured not to report referrers in this case, or if the navigation was initiated for a hyperlink with a `noreferrer` keyword.

Note: In the case of HTTP, the `referrer` DOM attribute will match the `Referer` (sic) header that was sent when fetching the current page.

Note: Typically user agents are configured to not report referrers in the case where the referrer uses an encrypted protocol and the current page does not (e.g. when navigating from an `https:` page to an `http:` page).

The **`cookie`** attribute must, on getting, return the same string as the value of the `Cookie` HTTP header it would include if fetching the resource indicated by the document's address over HTTP, as per RFC 2109 section 4.3.4. [RFC2109]

On setting, the `cookie` attribute must cause the user agent to act as it would when processing cookies if it had just attempted to fetch the document's address over HTTP, and had received a response with a `Set-Cookie` header whose value was the specified value, as per RFC 2109 sections 4.3.1, 4.3.2, and 4.3.3. [RFC2109]

Note: Since the `cookie` attribute is accessible across frames, the path restrictions on cookies are only a tool to help manage which cookies are sent to which parts of the site, and are not in any way a security feature.

The **`lastModified`** attribute, on getting, must return the date and time of the `Document`'s source file's last modification, in the user's local timezone, 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 digits in the range U+0030 DIGIT ZERO to U+0039 DIGIT NINE 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 filesystem for local files. If the last modification date and time are not known, the attribute must return the string `01/01/1970 00:00:00`.

The **`compatMode`** DOM attribute must return the literal string "`CSS1Compat`" unless the document has been set to **quirks mode** by the HTML parser, in which case it must instead return the literal string "`BackCompat`". The document can also be set to **limited quirks mode** (also known as "almost standards" mode). By default, the document is set to **no quirks mode** (also known as "standards mode").

As far as parsing goes, the quirks I know of are:

- Comment parsing is different.
- `p` can contain `table`
- Safari and IE have special parsing rules for <% ... %> (even in standards mode, though clearly this should be quirks-only).

Documents have an associated **character encoding**. When a `Document` object is created, the document's character encoding must be initialised to UTF-16. Various algorithms during page loading affect this value, as does the `charset` setter. [IANACHARSET]

The **`charset`** DOM attribute must, on getting, return the preferred MIME name of the document's character encoding. On setting, if the new value is an IANA-registered alias for a character encoding, the document's character encoding must be set to that character encoding. (Otherwise, nothing happens.)

The **`characterSet`** DOM attribute must, on getting, return the preferred MIME name of the document's character encoding.

The **`defaultCharset`** DOM attribute must, on getting, return the preferred MIME name of a character encoding, possibly the user's default encoding, or an encoding associated with the user's current geographical location, or any arbitrary encoding name.

Each document has a **current document readiness**. When a `Document` object is created, it must have its current document readiness set to the string "loading". Various algorithms during page loading affect this value. When the value is set, the user agent must fire a simple event called `readystatechanged` at the `Document` object.

The **`readyState`** DOM attribute must, on getting, return the current document readiness.

2.2 Elements

The nodes representing HTML elements in the DOM must implement, and expose to scripts, the interfaces listed for them in the relevant sections of this specification. This includes XHTML elements in XML documents, even when those documents are in another context (e.g. inside an XSLT transform).

The basic interface, from which all the HTML elements' interfaces inherit, and which must be used by elements that have no additional requirements, is the `HTMLElement` interface.

```
interface HTMLElement : Element {
  // DOM tree accessors
  NodeList getElementsByClassName(in DOMString classNames);

  // dynamic markup insertion
           attribute DOMString innerHTML;

  // metadata attributes
           attribute DOMString id;
           attribute DOMString title;
           attribute DOMString lang;
           attribute DOMString dir;
           attribute DOMString className;
  readonly attribute DOMTokenList classList;
  readonly attribute DOMStringMap dataset;

  // interaction
           attribute boolean irrelevant;
           attribute long tabIndex;
  void click();
  void focus();
  void blur();
  void scrollIntoView();
  void scrollIntoView(in boolean top);

  // commands
           attribute HTMLMenuElement contextMenu;

  // editing
           attribute boolean draggable;
           attribute DOMString contentEditable;
  readonly attribute DOMString isContentEditable;
  // styling
  readonly attribute CSSStyleDeclaration style;

  // data templates
           attribute DOMString template;
  readonly attribute HTMLDataTemplateElement templateElement;
           attribute DOMString ref;
  readonly attribute Node refNode;
           attribute DOMString registrationMark;
  readonly attribute DocumentFragment originalContent;

  // event handler DOM attributes
           attribute EventListener onabort;
           attribute EventListener onbeforeunload;
           attribute EventListener onblur;
           attribute EventListener onchange;
           attribute EventListener onclick;
           attribute EventListener oncontextmenu;
           attribute EventListener ondblclick;
           attribute EventListener ondrag;
           attribute EventListener ondragend;
           attribute EventListener ondragenter;
           attribute EventListener ondragleave;
           attribute EventListener ondragover;
           attribute EventListener ondragstart;
           attribute EventListener ondrop;
           attribute EventListener onerror;
           attribute EventListener onfocus;
           attribute EventListener onkeydown;
           attribute EventListener onkeypress;
           attribute EventListener onkeyup;
           attribute EventListener onload;
           attribute EventListener onmessage;
           attribute EventListener onmousedown;
           attribute EventListener onmousemove;
           attribute EventListener onmouseout;
           attribute EventListener onmouseover;
           attribute EventListener onmouseup;
           attribute EventListener onmousewheel;
           attribute EventListener onresize;
           attribute EventListener onscroll;
           attribute EventListener onselect;
           attribute EventListener onstorage;
           attribute EventListener onsubmit;
           attribute EventListener onunload;

};
```

As with the `HTMLDocument` interface, the `HTMLElement` interface holds methods and attributes related to a number of disparate features, and the members of this interface are therefore described in various different sections of this specification.

2.2.1 Reflecting content attributes in DOM attributes

Some DOM attributes are defined to **reflect** a particular content attribute. This means that on getting, the DOM attribute returns the current value of the content attribute, and on setting, the DOM attribute changes the value of the content attribute to the given value.

If a reflecting DOM attribute is a `DOMString` attribute whose content attribute is defined to contain a URI, then on getting, the DOM attribute must return the value of the content attribute, resolved to an absolute URI, and on setting, must set the content attribute to the specified literal value. If the content attribute is absent, the DOM attribute must return the default value, if the content attribute has one, or else the empty string.

If a reflecting DOM attribute is a `DOMString` attribute whose content attribute is defined to contain one or more URIs, then on getting, the DOM attribute must split the content attribute on spaces and return the concatenation of each token URI, resolved to an absolute URI, with a single U+0020 SPACE character between each URI; and on setting, must set the content attribute to the specified literal value. If the content attribute is absent, the DOM attribute must return the default value, if the content attribute has one, or else the empty string.

If a reflecting DOM attribute is a `DOMString` whose content attribute is an enumerated attribute, and the DOM attribute is **limited to only known values**, then, on getting, the DOM attribute must return the value associated with the state the attribute is in (in its canonical case), or the empty string if the attribute is in a state that has no associated keyword value; and on setting, if the new value case-insensitively matches one of the keywords given for that attribute, then the content attribute must be set to that value, otherwise, if the new value is the empty string, then the content attribute must be removed, otherwise, the setter must raise a `SYNTAX_ERR` exception.

If a reflecting DOM attribute is a `DOMString` but doesn't fall into any of the above categories, then the getting and setting must be done in a transparent, case-preserving manner.

If a reflecting DOM attribute is a boolean attribute, then the DOM attribute must return true if the attribute is set, and false if it is absent. On setting, the content attribute must be removed if the DOM attribute is set to false, and must be set to have the same value as its name if the DOM attribute is set to true. (This corresponds to the rules for boolean content attributes.)

If a reflecting DOM attribute is a signed integer type (`long`) then the content attribute must be parsed according to the rules for parsing signed integers first. If that fails, or if the attribute is absent, the default value must be returned instead, or 0 if there is no default value. On setting, the given value must be converted to a string representing the number as a valid integer in base ten and then that string must be used as the new content attribute value.

If a reflecting DOM attribute is an *unsigned* integer type (`unsigned long`) then the content attribute must be parsed according to the rules for parsing unsigned integers fi

Specification annotation system: Login...