



HTML 4.01 Specification

W3C Recommendation 24 December 1999

This version:

<http://www.w3.org/TR/1999/REC-html401-19991224>
 (plain text [794Kb], gzipped tar archive of HTML files [371Kb], a .zip archive of HTML files [405Kb], gzipped Postscript file [746Kb, 389 pages], gzipped PDF file [963Kb])

Latest version of HTML 4.01:

<http://www.w3.org/TR/html401>

Latest version of HTML 4:

<http://www.w3.org/TR/html4>

Latest version of HTML:

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

Previous version of HTML 4.01:

<http://www.w3.org/TR/1999/PR-html40-19990824>

Previous HTML 4 Recommendation:

<http://www.w3.org/TR/1998/REC-html40-19980424>

Editors:

Dave Raggett <dsr@w3.org>
 Arnaud Le Hors, W3C
 Ian Jacobs, W3C

Copyright ©1997-1999 W3C® (MIT, INRIA, Keio), All Rights Reserved. W3C liability, trademark, document use and software licensing rules apply.

Abstract

This specification defines the Hypertext Markup Language (HTML), the publishing language of the World Wide Web. This specification defines HTML 4.01, which is a subversion of HTML 4. In addition to the text, multimedia, and hyperlink features of the previous versions of HTML (HTML 3.2 [HTML32] [p.356] and HTML 2.0 [RFC1866] [p.356]), HTML 4 supports more multimedia options, scripting languages, style sheets, better printing facilities, and documents that are more accessible to users with disabilities. HTML 4 also takes great strides towards the internationalization of documents, with the goal of making the Web truly World Wide.

HTML 4 is an SGML application conforming to International Standard ISO 8879 -- Standard Generalized Markup Language [ISO8879] [p.353].

Status of this document

This section describes the status of this document at the time of its publication. Other documents may supersede this document. The latest status of this document series is maintained at the W3C.

This document specifies HTML 4.01, which is part of the HTML 4 line of specifications. The first version of HTML 4 was HTML 4.0 [HTML40] [p.353], published on 18 December 1997 and revised 24 April 1998. This specification is the first HTML 4.01 Recommendation. It includes non-editorial changes since the 24 April version of HTML 4.0 [p.312]. There have been some changes to the DTDs, for example. This document obsoletes previous versions of HTML 4.0, although W3C will continue to make those specifications and their DTDs available at the W3C Web site.

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

W3C recommends that user agents and authors (and in particular, authoring tools) produce HTML 4.01 documents rather than HTML 4.0 documents. W3C recommends that authors produce HTML 4 documents instead of HTML 3.2 documents. For reasons of backward compatibility, W3C also recommends that tools interpreting HTML 4 continue to support HTML 3.2 and HTML 2.0 as well.

For information about the next generation of HTML, "The Extensible HyperText Markup Language" [XHTML] [p.357], please refer to the W3C HTML Activity and the list of W3C Technical Reports.

This document has been produced as part of the W3C HTML Activity. The goals of the HTML Working Group (*Members only*) are discussed in the HTML Working Group charter (*Members only*).

A list of current W3C Recommendations and other technical documents can be found at <http://www.w3.org/TR>.

Public discussion on HTML features takes place on www-html@w3.org (archives of www-html@w3.org).

Available languages

The English version of this specification is the only normative version. However, for translations of this document, see <http://www.w3.org/MarkUp/html4-updates/translations>.

3. The <code>title</code> attribute	63	2. Unordered lists (<code>ul</code>), ordered lists (<code>ol</code>), and list items (<code>li</code>)	104
4. Meta data	64	3. Definition lists: the <code>dl</code> , <code>dt</code> , and <code>dd</code> elements	106
• Specifying meta data	64	1. Visual rendering of lists	108
• The <code>meta</code> element	65	4. The <code>dir</code> and <code>menu</code> elements	109
• Meta data profiles	68	11. Tables	111
5. The document body	69	1. Introduction to tables	111
1. The <code>body</code> element	69	2. Elements for constructing tables	113
2. Element identifiers: the <code>id</code> and <code>class</code> attributes	71	1. The <code>table</code> element	113
3. Block-level and inline elements	73	• Table directionality	115
4. Grouping elements: the <code>div</code> and <code>span</code> elements	73	2. Table Captions: The <code>caption</code> element	115
5. Headings: The <code>h1</code> , <code>h2</code> , <code>h3</code> , <code>h4</code> , <code>h5</code> , <code>h6</code> elements	75	3. Row groups: the <code>thead</code> , <code>tfoot</code> , and <code>tbody</code> elements	116
6. The <code>address</code> element	76	4. Column groups: the <code>colgroup</code> and <code>col</code> elements	118
8. Language information and text direction - <i>International considerations for text</i>	79	• The <code>colgroup</code> element	118
1. Specifying the language of content: the <code>lang</code> attribute	79	• The <code>col</code> element	120
1. Language codes	80	• Calculating the number of columns in a table	121
2. Inheritance of language codes	81	• Calculating the width of columns	122
3. Interpretation of language codes	81	5. Table rows: The <code>tr</code> element	124
2. Specifying the direction of text and tables: the <code>dir</code> attribute	82	6. Table cells: The <code>th</code> and <code>td</code> elements	125
1. Introduction to the bidirectional algorithm	82	• Cells that span several rows or columns	128
2. Inheritance of text direction information	83	3. Table formatting by visual user agents	130
3. Setting the direction of embedded text	84	1. Borders and rules	130
4. Overriding the bidirectional algorithm: the <code>bdo</code> element	85	2. Horizontal and vertical alignment	132
5. Character references for directionality and joining control	87	• Inheritance of alignment specifications	133
6. The effect of style sheets on bidirectionality	88	3. Cell margins	134
9. Text - <i>Paragraphs, Lines, and Phrases</i>	89	4. Table rendering by non-visual user agents	136
1. White space	89	1. Associating header information with data cells	136
2. Structured text	90	2. Categorizing cells	139
1. Phrase elements: <code>em</code> , <code>strong</code> , <code>dfn</code> , <code>code</code> , <code>samp</code> , <code>kbd</code> , <code>var</code> , <code>cite</code> , <code>abbr</code> , and <code>acronym</code>	90	3. Algorithm to find heading information	142
2. Quotations: The <code>blockquote</code> and <code>q</code> elements	92	5. Sample table	143
• Rendering quotations	93	12. Links - <i>Hypertext and Media-Independent Links</i>	145
3. Subscripts and superscripts: the <code>sub</code> and <code>sup</code> elements	94	1. Introduction to links and anchors	145
3. Lines and Paragraphs	94	1. Visiting a linked resource	145
1. Paragraphs: the <code>p</code> element	95	2. Other link relationships	147
2. Controlling line breaks	95	3. Specifying anchors and links	147
• Forcing a line break: the <code>br</code> element	96	4. Link titles	148
• Prohibiting a line break	96	5. Internationalization and links	148
3. Hyphenation	96	2. The <code>a</code> element	149
4. Preformatted text: The <code>pre</code> element	97	1. Syntax of anchor names	152
5. Visual rendering of paragraphs	98	2. Nested links are illegal	152
1. Paragraphs: the <code>p</code> element	98	3. Anchors with the <code>id</code> attribute	152
2. Controlling line breaks	99	4. Unavailable and unidentifiable resources	154
• Forcing a line break: the <code>br</code> element	99	3. Document relationships: the <code>link</code> element	154
• Prohibiting a line break	103	1. Forward and reverse links	155
10. Lists - <i>Unordered, Ordered, and Definition Lists</i>	103		
1. Introduction to lists	103		

2. Links and external style sheets	155	3. Floating objects	197
3. Links and search engines	155	• Float an object	197
4. Path information: the <code>BASE</code> element	157	• Float text around an object	198
1. Resolving relative URIs	158	2. Fonts	199
2. Resolving relative URIs	159	1. Font style elements: the <code>TT</code> , <code>I</code> , <code>B</code> , <code>BIG</code> , <code>SMALL</code> , <code>STRIKE</code> , <code>S</code> , and <code>U</code>	
13. Objects, Images, and Applets	159	elements	199
1. Introduction to objects, images, and applets	160	2. Font modifier elements: <code>FONT</code> and <code>BASEFONT</code>	200
2. Including an image: the <code>IMG</code> element	160	3. Rules: the <code>HR</code> element	202
3. Generic inclusion: the <code>OBJECT</code> element	162	16. Frames - <i>Multi-view presentation of documents</i>	205
1. Rules for rendering objects	165	1. Introduction to frames	205
2. Object initialization: the <code>PARAM</code> element	167	2. Layout of frames	206
3. Global naming schemes for objects	169	1. The <code>FRAMESET</code> element	206
4. Object declarations and instantiations	169	• Rows and columns	207
4. Including an applet: the <code>APPLET</code> element	171	• Nested frame sets	208
5. Notes on embedded documents	173	• Sharing data among frames	208
6. Image maps	173	2. The <code>FRAME</code> element	209
1. Client-side image maps: the <code>MAP</code> and <code>AREA</code> elements	174	• Setting the initial contents of a frame	210
• Client-side image map examples	176	• Visual rendering of a frame	212
2. Server-side image maps	179	3. Specifying target frame information	212
7. Visual presentation of images, objects, and applets	179	1. Setting the default target for links	213
1. Width and height	179	2. Target semantics	214
2. White space around images and objects	180	4. Alternate content	214
3. Borders	180	1. The <code>NOFRAMES</code> element	214
4. Alignment	180	2. Long descriptions of frames	215
8. How to specify alternate text	181	5. Inline frames: the <code>IFRAME</code> element	217
14. Style Sheets - <i>Adding style to HTML documents</i>	183	17. Forms - <i>User-input Forms: Text Fields, Buttons, Menus, and more</i>	219
1. Introduction to style sheets	183	1. Introduction to forms	219
2. Adding style to HTML	185	2. Controls	220
1. Setting the default style sheet language	186	1. Control types	221
2. Inline style information	186	3. The <code>FORM</code> element	222
3. Header style information: the <code>STYLE</code> element	187	4. The <code>INPUT</code> element	224
4. Media types	189	1. Control types created with <code>INPUT</code>	226
3. External style sheets	190	2. Examples of forms containing <code>INPUT</code> controls	227
1. Preferred and alternate style sheets	190	5. The <code>BUTTON</code> element	228
2. Specifying external style sheets	191	6. The <code>SELECT</code> , <code>OPTGROUP</code> , and <code>OPTION</code> elements	230
4. Cascading style sheets	192	1. Pre-selected options	231
1. Media-dependent cascades	192	7. The <code>TEXTAREA</code> element	234
2. Inheritance and cascading	193	8. The <code>ISINDEX</code> element	236
5. Hiding style data from user agents	193	9. Labels	237
6. Linking to style sheets with HTTP headers	194	1. The <code>LABEL</code> element	237
15. Alignment, font styles, and horizontal rules	195	10. Adding structure to forms: the <code>FIELDSET</code> and <code>LEGEND</code> elements	239
1. Formatting	195	11. Giving focus to an element	241
1. Background color	195	1. Tabbing navigation	241
2. Alignment	195		

2. Access keys	242
12. Disabled and read-only controls	243
1. Disabled controls	244
2. Read-only controls	244
13. Form submission	245
1. Form submission method	245
2. Successful controls	245
3. Processing form data	246
• Step one: Identify the successful controls	246
• Step two: Build a form data set	246
• Step three: Encode the form data set	246
• Step four: Submit the encoded form data set	247
4. Form content types	247
• application/x-www-form-urlencoded	247
• multipart/form-data	248
18. Scripts - <i>Animated Documents and Smart Forms</i>	251
1. Introduction to scripts	251
2. Designing documents for user agents that support scripting	252
1. The SCRIPT element	252
2. Specifying the scripting language	253
• The default scripting language	253
• Local declaration of a scripting language	254
• References to HTML elements from a script	254
3. Intrinsic events	254
4. Dynamic modification of documents	258
3. Designing documents for user agents that don't support scripting	258
1. The NOSCRIPT element	258
2. Hiding script data from user agents	259
19. SGML reference information for HTML - <i>Formal definition of HTML and validation</i>	261
1. Document Validation	261
2. Sample SGML catalog	262
20. SGML Declaration of HTML 4	263
1. SGML Declaration	263
2. Document Type Definition	265
22. Transitional Document Type Definition	279
23. Frameset Document Type Definition	297
24. Character entity references in HTML 4	299
1. Introduction to character entity references	299
2. Character entity references for ISO 8859-1 characters	299
1. The list of characters	300
3. Character entity references for symbols, mathematical symbols, and Greek letters	303
1. The list of characters	304
4. Character entity references for markup-significant and internationalization characters	308
1. The list of characters	308
A. Changes	311
1. Changes between 24 April 1998 HTML 4.0 and 24 December 1999 HTML 4.01 versions	312
1. Changes to the specification	312
• General changes	312
• On SGML and HTML	312
• HTML Document Representation	312
• Basic HTML data types	312
• Global structure of an HTML document	313
• Language information and text direction	313
• Tables	313
• Links	313
• Objects, Images, and Applets	314
• Style Sheets in HTML Documents	314
• Frames	314
• Forms	315
• SGML Declaration	315
• Strict DTD	315
• Notes	315
• References	316
2. Errors that were corrected	316
3. Minor typographical errors that were corrected	318
4. Clarifications	322
5. Known Browser problems	322
2. Changes between 18 December 1997 and 24 April 1998 versions	322
1. Errors that were corrected	323
2. Minor typographical errors that were corrected	325
3. Changes between HTML 3.2 and HTML 4.0 (18 December 1997)	327
1. Changes to elements	327
• New elements	327
• Deprecated elements	327
• Obsolete elements	328
2. Changes to attributes	328
3. Changes for accessibility	328
4. Changes for meta data	328
5. Changes for text	328
6. Changes for links	328
7. Changes for tables	328
8. Changes for images, objects, and image maps	329

9. Changes for forms	330
10. Changes for style sheets	330
11. Changes for frames	330
12. Changes for scripting	330
13. Changes for internationalization	330
B. Performance, Implementation, and Design Notes	333
1. Notes on invalid documents	334
2. Special characters in URI attribute values	334
1. Non-ASCII characters in URI attribute values	334
2. Ampersands in URI attribute values	335
3. SGML implementation notes	335
1. Line breaks	335
2. Specifying non-HTML data	336
• Element content	336
• Attribute values	337
3. SGML features with limited support	337
4. Boolean attributes	337
5. Marked Sections	338
6. Processing Instructions	338
7. Shorthand markup	338
4. Notes on helping search engines index your Web site	339
1. Search robots	340
• The robots.txt file	340
• Robots and the META element	341
2. Notes on tables	342
1. Design rationale	342
• Dynamic reformatting	342
• Incremental display	342
• Structure and presentation	343
• Row and column groups	344
• Accessibility	344
2. Recommended Layout Algorithms	344
• Fixed Layout Algorithm	345
• Autolayout Algorithm	345
6. Notes on forms	347
1. Incremental display	347
2. Future projects	348
7. Notes on scripting	348
1. Reserved syntax for future script macros	348
• Current Practice for Script Macros	348
8. Notes on frames	350
9. Notes on accessibility	350
10. Notes on security	350
1. Security issues for forms	350
• References	353
1. Normative references	353
2. Informative references	355
• Index of Elements	359
• Index of Attributes	363
• Index	375

1 About the HTML 4 Specification

Contents

1. How the specification is organized	15
2. Document conventions	16
1. Elements and attributes	17
2. Notes and examples	17
3. Acknowledgments	17
1. Acknowledgments for the current revision	18
4. Copyright Notice	18

1.1 How the specification is organized

This specification is divided into the following sections:

Sections 2 and 3: Introduction to HTML 4

The introduction describes HTML's place in the scheme of the World Wide Web, provides a brief history of the development of HTML, highlights what can be done with HTML 4, and provides some HTML authoring tips.

The brief SGML tutorial gives readers some understanding of HTML's relationship to SGML and gives summary information on how to read the HTML Document Type Definition (DTD).

Sections 4 - 24: HTML 4 reference manual

The bulk of the reference manual consists of the HTML language reference, which defines all elements and attributes of the language.

This document has been organized by topic rather than by the grammar of HTML. Topics are grouped into three categories: structure, presentation, and interactivity. Although it is not easy to divide HTML constructs perfectly into these three categories, the model reflects the HTML Working Group's experience that separating a document's structure from its presentation produces more effective and maintainable documents.

The language reference consists of the following information:

- What characters [p.41] may appear in an HTML document.
- Basic data types [p.49] of an HTML document.
- Elements that govern the structure of an HTML document, including text [p.89], lists [p.103], tables [p.111], links [p.145], and included objects, images, and applets [p.159].
- Elements that govern the presentation of an HTML document, including style sheets [p.183], fonts, colors, rules, and other visual presentation [p.195], and frames for multi-windowed presentations [p.205].

- Elements that govern interactivity with an HTML document, including forms for user input [p.219] and scripts for active documents [p.251].
- The SGML formal definition of HTML:
 - The SGML declaration of HTML [p.263].
 - Three DTDs: strict [p.265], transitional [p.279], and frameset [p.297].
 - The list of character references [p.299].

Appendixes

The first appendix contains information about changes from HTML 3.2 [p.311] to help authors and implementors with the transition to HTML 4, and changes from the 18 December 1997 specification [p.322]. The second appendix contains performance and implementation notes [p.333], and is primarily intended to help implementors create user agents for HTML 4.

References

A list of normative and informative references.

Indexes

Three indexes give readers rapid access to the definition of key concepts [p.375], elements [p.359] and attributes [p.363].

1.2 Document conventions

This document has been written with two types of readers in mind: authors and implementors. We hope the specification will provide authors with the tools they need to write efficient, attractive, and accessible documents, without over-exposing them to HTML's implementation details. Implementors, however, should find all they need to build conforming user agents.

The specification may be approached in several ways:

- **Read from beginning to end.** The specification begins with a general presentation of HTML and becomes more and more technical and specific towards the end.
- **Quick access to information.** In order to get information about syntax and semantics as quickly as possible, the online version of the specification includes the following features:
 1. Every reference to an element or attribute is linked to its definition in the specification. Each element or attribute is defined in only one location.
 2. Every page includes links to the indexes, so you never are more than two links away from finding the definition of an element [p.359] or attribute [p.363].
 3. The front pages of each section of the language reference manual extend the initial table of contents with more detail about that section.

1.2.1 Elements and attributes

Element names are written in uppercase letters (e.g., BODY). Attribute names are written in lowercase letters (e.g., lang, onsubmit). Recall that in HTML, element and attribute names are case-insensitive; the convention is meant to encourage readability.

Element and attribute names in this document have been marked up and may be rendered specially by some user agents.

Each attribute definition specifies the type of its value. If the type allows a small set of possible values, the definition lists the set of values, separated by a bar (|).

After the type information, each attribute definition indicates the case-sensitivity of its values, between square brackets ("[]"). See the section on case information [p.49] for details.

1.2.2 Notes and examples

Informative notes are emphasized to stand out from surrounding text and may be rendered specially by some user agents.

All examples illustrating deprecated [p.38] usage are marked as "DEPRECATED EXAMPLE". Deprecated examples also include recommended alternate solutions. All examples that illustrates illegal usage are clearly marked "ILLEGAL EXAMPLE".

Examples and notes have been marked up and may be rendered specially by some user agents.

1.3 Acknowledgments

Thanks to everyone who has helped to author the working drafts that went into the HTML 4 specification, and to all those who have sent suggestions and corrections.

Many thanks to the Web Accessibility Initiative task force (WAI HC group) for their work on improving the accessibility of HTML and to T. V. Raman (Adobe) for his early work on developing accessible forms.

The authors of this specification, the members of the W3C HTML Working Group, deserve much applause for their diligent review of this document, their constructive comments, and their hard work: John D. Burger (MITRE), Steve Byrne (JavaSoft), Martin J. Dürst (University of Zurich), Daniel Glazman (Electricité de France), Scott Isaacs (Microsoft), Murray Maloney (GRIF), Steven Pemberton (CWI), Robert Parnett (Lotus), Jared Sorensen (Novell), Powell Smith (IBM), Robert Stevahn (HP), Ed Tecon (Microsoft), Jeffrey Veen (HotWired), Mike Wexler (Adobe), Misha Wolf (Reuters), and Lauren Wood (SoftQuad).

Thank you Dan Connolly (W3C) for rigorous and bountiful input as part-time editor and thoughtful guidance as chairman of the HTML Working Group. Thank you Sally Khudairi (W3C) for your indispensable work on press releases.

Thanks to David M. Abrahamson and Roger Price for their careful reading of the specification and constructive comments.

Thanks to Jan Kärman, author of htm12ps for helping so much in creating the Postscript version of the specification.

Of particular help from the W3C at Sophia-Antipolis were Janet Bertot, Bert Bos, Stephane Boyera, Daniel Dardailler, Yves Lafon, Håkon Lie, Chris Lilley, and Colas Nahaboo (Bull).

Lastly, thanks to Tim Berners-Lee without whom none of this would have been possible.

1.3.1 Acknowledgments for the current revision

Many thanks to Shane McCarron for tracking errata for this revision of the specification.

1.4 Copyright Notice

For information about copyrights, please refer to the W3C Intellectual Property Notice, the W3C Document Notice, and the W3C IPR Software Notice.

2 Introduction to HTML 4

Contents

1. What is the World Wide Web?	19
1. Introduction to URIs	19
2. Fragment identifiers	20
3. Relative URIs	20
2. What is HTML?	21
1. A brief history of HTML	21
3. HTML 4	22
1. Internationalization	22
2. Accessibility	22
3. Tables	23
4. Compound documents	23
5. Style sheets	24
6. Scripting	24
7. Printing	24
4. Authoring documents with HTML 4	24
1. Separate structure and presentation	25
2. Consider universal accessibility to the Web	25
3. Help user agents with incremental rendering	25

2.1 What is the World Wide Web?

The *World Wide Web (Web)* is a network of information resources. The Web relies on three mechanisms to make these resources readily available to the widest possible audience:

1. A uniform naming scheme for locating resources on the Web (e.g., URIs).
2. Protocols, for access to named resources over the Web (e.g., HTTP).
3. Hypertext, for easy navigation among resources (e.g., HTML).

The ties between the three mechanisms are apparent throughout this specification.

2.1.1 Introduction to URIs

Every resource available on the Web -- HTML document, image, video clip, program, etc. -- has an address that may be encoded by a *Universal Resource Identifier*, or "URI".

URIs typically consist of three pieces:

1. The naming scheme of the mechanism used to access the resource.
2. The name of the machine hosting the resource.
3. The name of the resource itself, given as a path.

Consider the URI that designates the W3C Technical Reports page:

```
http://www.w3.org/TR
```

This URI may be read as follows: There is a document available via the HTTP protocol (see [RFC2616] [p.354]), residing on the machine `www.w3.org`, accessible via the path `/TR`. Other schemes you may see in HTML documents include "mailto" for email and "ftp" for FTP.

Here is another example of a URI. This one refers to a user's mailbox:

```
...this is text...
For all comments, please send email to
<A href="mailto:Joe@someplace.com">Joe Cool</A>.
```

Note. Most readers may be familiar with the term "URL" and not the term "URI". URIs form a subset of the more general URI naming scheme.

2.1.2 Fragment identifiers

Some URIs refer to a location within a resource. This kind of URI ends with "#" followed by an anchor identifier (called the *fragment identifier*). For instance, here is a URI pointing to an anchor named `section_2`:

```
http://someite.com/html/top.html#section_2
```

2.1.3 Relative URIs

A *relative URI* doesn't contain any naming scheme information. Its path generally refers to a resource on the same machine as the current document. Relative URIs may contain relative path components (e.g., ".") means one level up in the hierarchy defined by the path), and may contain fragment identifiers [p.20].

Relative URIs are resolved to full URIs [p.158] using a base URI. As an example of relative URI resolution, assume we have the base URI `"http://www.acme.com/support/intro.html"`. The relative URI in the following markup for a hypertext link:

```
<A href="suppliers.html">Suppliers</A>
```

would expand to the full URI `"http://www.acme.com/support/suppliers.html"`, while the relative URI in the following markup for an image

```
<IMG src="..icons/logo.gif" alt="logo">
```

would expand to the full URI `"http://www.acme.com/icons/logo.gif"`.

In HTML, URIs are used to:

- Link to another document or resource, (see the `A` and `LINK` elements).
- Link to an external style sheet or script (see the `LINK` and `SCRIPT` elements).
- Include an image, object, or applet in a page, (see the `IMG`, `OBJECT`, `APPLET` and `INPUT` elements).
- Create an image map (see the `MAP` and `AREA` elements).
- Submit a form (see `FORM`).
- Create a frame document (see the `FRAME` and `IFRAME` elements).
- Cite an external reference (see the `Q`, `BLOCKQUOTE`, `INS` and `DEL` elements).
- Refer to metadata conventions describing a document (see the `HEAD` element).

Please consult the section on the URI [p.51] type for more information about URIs.

2.2 What is HTML?

To publish information for global distribution, one needs a universally understood language, a kind of publishing mother tongue that all computers may potentially understand. The publishing language used by the World Wide Web is HTML (from HyperText Markup Language).

HTML gives authors the means to:

- Publish online documents with headings, text, tables, lists, photos, etc.
- Retrieve online information via hypertext links, at the click of a button.
- Design forms for conducting transactions with remote services, for use in searching for information, making reservations, ordering products, etc.
- Include spread-sheets, video clips, sound clips, and other applications directly in their documents.

2.2.1 A brief history of HTML

HTML was originally developed by Tim Berners-Lee while at CERN, and popularized by the Mosaic browser developed at NCSA. During the course of the 1990s it has blossomed with the explosive growth of the Web. During this time, HTML has been extended in a number of ways. The Web depends on Web page authors and vendors sharing the same conventions for HTML. This has motivated joint work on specifications for HTML.

HTML 2.0 (November 1995, see [RFC:1866] [p.356]) was developed under the aegis of the Internet Engineering Task Force (IETF) to codify common practice in late 1994. HTML+ (1993) and HTML 3.0 (1995, see [HTML:30] [p.355]) proposed much richer versions of HTML. Despite never receiving consensus in standards discussions, these drafts led to the adoption of a range of new features. The efforts of the World Wide Web Consortium's HTML Working Group to codify common practice in 1996 resulted in HTML 3.2 (January 1997, see [HTML:32] [p.356]). Changes from HTML 3.2 are summarized in Appendix A [p.311]

Most people agree that HTML documents should work well across different browsers and platforms. Achieving interoperability lowers costs to content providers since they must develop only one version of a document. If the effort is not made, there is much greater risk that the Web will devolve into a proprietary world of incompatible formats, ultimately reducing the Web's commercial potential for all participants.

Each version of HTML has attempted to reflect greater consensus among industry players so that the investment made by content providers will not be wasted and that their documents will not become unreadable in a short period of time.

HTML has been developed with the vision that all manner of devices should be able to use information on the Web. PCs with graphics displays of varying resolution and color depths, cellular telephones, hand held devices, devices for speech for output and input, computers with high or low bandwidth, and so on.

2.3 HTML 4

HTML 4 extends HTML with mechanisms for style sheets, scripting, frames, embedding objects, improved support for right to left and mixed direction text, richer tables, and enhancements to forms, offering improved accessibility for people with disabilities.

HTML 4.01 is a revision of HTML 4.0 that corrects errors and makes some changes since the previous revision. [p.311]

2.3.1 Internationalization

This version of HTML has been designed with the help of experts in the field of internationalization, so that documents may be written in every language and be transported easily around the world. This has been accomplished by incorporating [RFC2070] [p.356], which deals with the internationalization of HTML.

One important step has been the adoption of the ISO/IEC:10646 standard (see [ISO10646] [p.353]) as the document character set for HTML. This is the world's most inclusive standard dealing with issues of the representation of international characters, text direction, punctuation, and other world language issues.

HTML now offers greater support for diverse human languages within a document. This allows for more effective indexing of documents for search engines, higher-quality typography, better text-to-speech conversion, better hyphenation, etc.

2.3.2 Accessibility

As the Web community grows and its members diversify in their abilities and skills, it is crucial that the underlying technologies be appropriate to their specific needs. HTML has been designed to make Web pages more accessible to those with physical limitations. HTML 4 developments inspired by concerns for accessibility include:

- Better distinction between document structure and presentation, thus encouraging the use of style sheets instead of HTML presentation elements and attributes.
- Better forms, including the addition of access keys, the ability to group form controls semantically, the ability to group `SELECT` options semantically, and active labels.
- The ability to markup a text description of an included object (with the `OBJECT` element).
- A new client-side image map mechanism (the `MAP` element) that allows authors to integrate image and text links.
- The requirement that alternate text accompany images included with the `IMG` element and image maps included with the `AREA` element.
- Support for the `title` and `lang` attributes on all elements.
- Support for the `ABBR` and `ACRONYM` elements.
- A wider range of target media (tty, braille, etc.) for use with style sheets.
- Better tables, including captions, column groups, and mechanisms to facilitate non-visual rendering.
- Long descriptions of tables, images, frames, etc.

Authors who design pages with accessibility issues in mind will not only receive the blessings of the accessibility community, but will benefit in other ways as well: well-designed HTML documents that distinguish structure and presentation will adapt more easily to new technologies.

Note. For more information about designing accessible HTML documents, please consult [WA1] [p.357].

2.3.3 Tables

The new table model in HTML is based on [RFC1942] [p.356]. Authors now have greater control over structure and layout (e.g., column groups). The ability of designers to recommend column widths allows user agents to display table data incrementally (as it arrives) rather than waiting for the entire table before rendering.

Note. At the time of writing, some HTML authoring tools rely extensively on tables for formatting, which may easily cause accessibility problems.

2.3.4 Compound documents

HTML now offers a standard mechanism for embedding generic media objects and applications in HTML documents. The `OBJECT` element (together with its more specific ancestor elements `IMG` and `APPLET`) provides a mechanism for including images, video, sound, mathematics, specialized applications, and other objects in a document. It also allows authors to specify a hierarchy of alternate renderings for user agents that don't support a specific rendering.

2.3.5 Style sheets

Style sheets simplify HTML markup and largely relieve HTML of the responsibilities of presentation. They give both authors and users control over the presentation of documents -- font information, alignment, colors, etc.

Style information can be specified for individual elements or groups of elements. Style information may be specified in an HTML document or in external style sheets.

The mechanisms for associating a style sheet with a document is independent of the style sheet language.

Before the advent of style sheets, authors had limited control over rendering. HTML 3.2 included a number of attributes and elements offering control over alignment, font size, and text color. Authors also exploited tables and images as a means for laying out pages. The relatively long time it takes for users to upgrade their browsers means that these features will continue to be used for some time. However, since style sheets offer more powerful presentation mechanisms, the World Wide Web Consortium will eventually phase out many of HTML's presentation elements and attributes. Throughout the specification elements and attributes at risk are marked as "deprecated [p.38]". They are accompanied by examples of how to achieve the same effects with other elements or style sheets.

2.3.6 Scripting

Through scripts, authors may create dynamic Web pages (e.g., "smart forms" that react as users fill them out) and use HTML as a means to build networked applications.

The mechanisms provided to include scripts in an HTML document are independent of the scripting language.

2.3.7 Printing

Sometimes, authors will want to make it easy for users to print more than just the current document. When documents form part of a larger work, the relationships between them can be described using the `HTML_LINK` element or using W3C's Resource Description Framework (RDF) (see [RDF10] [p.356]).

2.4 Authoring documents with HTML 4

We recommend that authors and implementors observe the following general principles when working with HTML 4.

2.4.1 Separate structure and presentation

HTML has its roots in SGML, which has always been a language for the specification of structural markup. As HTML matures, more and more of its presentational elements and attributes are being replaced by other mechanisms, in particular style sheets. Experience has shown that separating the structure of a document from its presentational aspects reduces the cost of serving a wide range of platforms, media, etc., and facilitates document revisions.

2.4.2 Consider universal accessibility to the Web

To make the Web more accessible to everyone, notably those with disabilities, authors should consider how their documents may be rendered on a variety of platforms: speech-based browsers, braille-readers, etc. We do not recommend that authors limit their creativity, only that they consider alternate renderings in their design. HTML offers a number of mechanisms to this end (e.g., the `alt` attribute, the `accesskey` attribute, etc.)

Furthermore, authors should keep in mind that their documents may be reaching a far-off audience with different computer configurations. In order for documents to be interpreted correctly, authors should include in their documents information about the natural language and direction of the text, how the document is encoded, and other issues related to internationalization.

2.4.3 Help user agents with incremental rendering

By carefully designing their tables and making use of new table features in HTML 4, authors can help user agents render documents more quickly. Authors can learn how to design tables for incremental rendering (see the `TABLE` element). Implementors should consult the notes on tables [p.342] in the appendix for information on incremental algorithms.

3 On SGML and HTML

Contents

1. Introduction to SGML	27
2. SGML constructs used in HTML	28
1. Elements	28
2. Attributes	29
3. Character references	30
4. Comments	30
3. How to read the HTML DTD	31
1. DTD Comments	31
2. Parameter entity definitions	31
3. Element declarations	32
• Content model definitions	33
4. Attribute declarations	34
• DTD entities in attribute definitions	35
• Boolean attributes	36

This section of the document introduces SGML and discusses its relationship to HTML. A complete discussion of SGML is left to the standard (see [ISO8879] [p.353]).

3.1 Introduction to SGML

SGML is a system for defining markup languages. Authors *mark up* their documents by representing structural, presentational, and semantic information alongside content. HTML is one example of a markup language. Here is an example of an HTML document:

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01//EN"
" http://www.w3.org/TR/html4/strict.dtd">
<HTML>
<HEAD>
<TITLE>My first HTML document</TITLE>
</HEAD>
<BODY>
<P>Hello world!
</BODY>
</HTML>
```

An HTML document is divided into a head section (here, between <HEAD> and </HEAD>) and a body (here, between <BODY> and </BODY>). The title of the document appears in the head (along with other information about the document), and the content of the document appears in the body. The body in this example contains just one paragraph, marked up with <P>.

Each markup language defined in SGML is called an *SGML application*. An SGML application is generally characterized by:

1. An SGML declaration [p.263]. The *SGML declaration* specifies which characters and delimiters may appear in the application.
2. A *document type definition (DTD)* [p.265]. The DTD defines the syntax of markup constructs. The DTD may include additional definitions such as character entity references [p.30].
3. A specification that describes the semantics to be ascribed to the markup. This specification also imposes syntax restrictions that cannot be expressed within the DTD.
4. Document instances containing data (content) and markup. Each instance contains a reference to the DTD to be used to interpret it.

This specification includes an SGML declaration [p.263], three document type definitions (see the section on HTML version information [p.59] for a description of the three), and a list of character references [p.30].

3.2 SGML constructs used in HTML

The following sections introduce SGML constructs that are used in HTML.

The appendix lists some SGML features [p.337] that are not widely supported by HTML tools and user agents and should be avoided.

3.2.1 Elements

An SGML document type definition [p.265] declares *element types* that represent structures or desired behavior. HTML includes element types that represent paragraphs, hypertext links, lists, tables, images, etc.

Each *element type declaration* generally describes three parts: a start tag, content, and an end tag.

The element's name appears in the *start tag* (written <element-name>) and the *end tag* (written </element-name>); note the slash before the element name in the end tag. For example, the start and end tags of the UL element type delimit the items in a list:

```
<UL>
<LI><P>...list item 1...
<LI><P>...list item 2...
</UL>
```

Some HTML element types allow authors to omit end tags (e.g., the P and LI element types). A few element types also allow the start tags to be omitted; for example, HEAD and BODY. The HTML DTD indicates for each element type whether the start tag and end tag are required.

Some HTML element types have no content. For example, the line break element `BR` has no content; its only role is to terminate a line of text. Such *empty* elements never have end tags. The document type definition [p.265] and the text of the specification indicate whether an element type is empty (has no content) or, if it can have content, what is considered legal content.

Element names are always case-insensitive.

Please consult the SGML standard for information about rules governing elements (e.g., they must be properly nested, an end tag closes, back to the matching start tag, all unclosed intervening start tags with omitted end tags (section 7.5.1), etc.).

For example, the following paragraph:

```
<P>This is the first paragraph.</P>
...a block element...
```

may be rewritten without its end tag:

```
<P>This is the first paragraph.
...a block element...
```

since the `<P>` start tag is closed by the following block element. Similarly, if a paragraph is enclosed by a block element, as in:

```
<DIV>
<P>This is the paragraph.
</DIV>
```

the end tag of the enclosing block element (here, `<DIV>`) implies the end tag of the open `<P>` start tag.

Elements are *not* tags. Some people refer to elements as tags (e.g., "the *P tag*"). Remember that the element is one thing, and the tag (be it start or end tag) is another. For instance, the *HEAD* element is always present, even though both start and end *HEAD* tags may be missing in the markup.

All the element types declared in this specification are listed in the element index [p.359].

3.2.2 Attributes

Elements may have associated properties, called *attributes*, which may have values (by default, or set by authors or scripts). Attribute/value pairs appear before the final `>` of an element's start tag. Any number of (legal) attribute value pairs, separated by spaces, may appear in an element's start tag. They may appear in any order.

In this example, the `id` attribute is set for an *H1* element:

```
<H1 id="section1" >
This is an identified heading thanks to the id attribute
</H1>
```

By default, SGML requires that all attribute values be delimited using either double quotation marks (ASCII decimal 34) or single quotation marks (ASCII decimal 39). Single quote marks can be included within the attribute value when the value is delimited by double quote marks, and vice versa. Authors may also use numeric character references [p.30] to represent double quotes (`"`) and single quotes (`'`). For double quotes authors can also use the character entity reference [p.30] `"`.

In certain cases, authors may specify the value of an attribute without any quotation marks. The attribute value may only contain letters (a-z and A-Z), digits (0-9), hyphens (ASCII decimal 45), periods (ASCII decimal 46), underscores (ASCII decimal 95), and colons (ASCII decimal 58). We recommend using quotation marks even when it is possible to eliminate them.

Attribute names are always case-insensitive.

Attribute values are generally case-insensitive. The definition of each attribute in the reference manual indicates whether its value is case-insensitive.

All the attributes defined by this specification are listed in the attribute index [p.363].

3.2.3 Character references

Character references are numeric or symbolic names for characters that may be included in an HTML document. They are useful for referring to rarely used characters, or those that authoring tools make it difficult or impossible to enter. You will see character references throughout this document; they begin with a `&` sign and end with a semi-colon (`;`). Some common examples include:

- `<` represents the `<` sign.
- `>` represents the `>` sign.
- `"` represents the `"` mark.
- `å` (in decimal) represents the letter "a" with a small circle above it.
- `И` (in decimal) represents the Cyrillic capital letter "I".
- `水` (in hexadecimal) represents the Chinese character for water.

We discuss HTML character references [p.45] in detail later in the section on the HTML document character set [p.41]. The specification also contains a list of character references [p.299] that may appear in HTML 4 documents.

3.2.4 Comments

HTML comments have the following syntax:

```
<!-- this is a comment -->
<!-- and so is this one,
      which occupies more than one line -->
```

White space is not permitted between the markup declaration open delimiter("<") and the comment open delimiter ("--"), but is permitted between the comment close delimiter ("--") and the markup declaration close delimiter (">"). A common error is to include a string of hyphens ("---") within a comment. Authors should avoid putting two or more adjacent hyphens inside comments.

Information that appears between comments has no special meaning (e.g., character references [p.30] are not interpreted as such).

Note that comments are markup.

3.3 How to read the HTML DTD

Each element and attribute declaration in this specification is accompanied by its document type definition [p.265] fragment. We have chosen to include the DTD fragments in the specification rather than seek a more approachable, but longer and less precise means of describing an element's properties. The following tutorial should allow readers unfamiliar with SGML to read the DTD and understand the technical details of the HTML specification.

3.3.1 DTD Comments

In DTDs, comments may spread over one or more lines. In the DTD, comments are delimited by a pair of "--" marks, e.g.

```
<ELEMENT PARAM - O EMPTY -- named property value -->
```

Here, the comment "named property value" explains the use of the PARAM element type. Comments in the DTD are informative only.

3.3.2 Parameter entity definitions

The HTML DTD [p.265] begins with a series of parameter entity definitions. A *parameter entity definition* defines a kind of macro that may be referenced and expanded elsewhere in the DTD. These macros may not appear in HTML documents, only in the DTD. Other types of macros, called character references [p.30], may be used in the text of an HTML document or within attribute values.

When the parameter entity is referred to by name in the DTD, it is expanded into a string.

A parameter entity definition begins with the keyword <ENTITY % followed by the entity name, the quoted string the entity expands to, and finally a closing >. Instances of parameter entities in a DTD begin with "%", then the parameter entity name, and terminated by an optional ";".

The following example defines the string that the "%fontstyle;" entity will expand to.

```
<ENTITY % fontstyle "TT | I | B | BIG | SMALL">
```

The string the parameter entity expands to may contain other parameter entity names. These names are expanded recursively. In the following example, the "%inline;" parameter entity is defined to include the "%fontstyle;", "%phrase;", "%special;" and "%formctrl;" parameter entities.

```
<ENTITY % inline "#PCDATA | %fontstyle; | %phrase; | %special; | %formctrl;">
```

You will encounter two DTD entities frequently in the HTML DTD [p.265]: "%block;" and "%inline;". They are used when the content model includes block-level and inline elements [p.73], respectively (defined in the section on the global structure of an HTML document [p.59]).

3.3.3 Element declarations

The bulk of the HTML DTD consists of the declarations of *element types* and their attributes. The <ELEMENT keyword begins a declaration and the > character ends it. Between these are specified:

1. The element's name.
2. Whether the element's tags are optional. Two hyphens that appear after the element name mean that the start and end tags are mandatory. One hyphen followed by the letter "O" indicates that the end tag can be omitted. A pair of letter "O"s indicate that both the start and end tags can be omitted.
3. The element's content, if any. The allowed content for an element is called its *content model*. Element types that are designed to have no content are called *empty elements*. The content model for such element types is declared using the keyword "EMPTY".

In this example:

```
<ELEMENT UL - - (LI)+>
```

- The element type being declared is **UL**.
- The two hyphens indicate that both the start tag and the end tag for this element type are required.
- The content model for this element type is declared to be "at least one LI element". Below, we explain how to specify content models.

This example illustrates the declaration of an empty element type:

```
<ELEMENT IMG - O EMPTY>
```

- The element type being declared is **IMG**.
- The hyphen and the following "O" indicate that the end tag can be omitted, but together with the content model "EMPTY", this is strengthened to the rule that the end tag **must** be omitted.
- The "EMPTY" keyword means that instances of this type must not have content.

Content model definitions

The content model describes what may be contained by an instance of an element type. Content model definitions may include:

- The names of allowed or forbidden element types (e.g., the `UL` element contains instances of the `LI` element type, and the `P` element type may not contain other `P` elements).
- DTD entities (e.g., the `LABEL` element contains instances of the `"%inline;"` parameter entity).
- Document text (indicated by the SGML construct `"#PCDATA"`). Text may contain character references [p.45]. Recall that these begin with `&` and end with a semicolon (e.g., `"Herg´s adventures of Tintin"` contains the character entity reference for the "e acute" character).

The content model of an element is specified with the following syntax. Please note that the list below is a simplification of the full SGML syntax rules and does not address, e.g., precedences.

(...)
Delimits a group.

A A must occur, one time only.

A+ A must occur one or more times.

A? A must occur zero or one time.

A* A may occur zero or more times.

+(A) A may occur.

-(A) A must not occur.

A | B Either A or B must occur, but not both.

A , B Both A and B must occur, in that order.

A & B Both A and B must occur, in any order.

Here are some examples from the HTML DTD:

```
<!ELEMENT UL - - (LI)+>
```

The `UL` element must contain one or more `LI` elements.

```
<!ELEMENT DL - - (DT|DD)+>
```

The `DL` element must contain one or more `DT` or `DD` elements in any order.

```
<!ELEMENT OPTION - O (#PCDATA)>
```

The `OPTION` element may only contain text and entities, such as `&`; -- this is indicated by the SGML data type `#PCDATA`.

A few HTML element types use an additional SGML feature to exclude elements from their content model. Excluded elements are preceded by a hyphen. Explicit exclusions override permitted elements.

In this example, the `-(A)` signifies that the element `A` cannot appear in another `A` element (i.e., anchors may not be nested).

```
<!ELEMENT A - - (%inline;)* -(A)>
```

Note that the `A` element type is part of the DTD parameter entity `"%inline;"`, but is excluded explicitly because of `-(A)`.

Similarly, the following element type declaration for `FORM` prohibits nested forms:

```
<!ELEMENT FORM - - (%block;|SCRIPT)* -(FORM)>
```

3.3.4 Attribute declarations

The `<!ATTLIST` keyword begins the declaration of attributes that an element may take. It is followed by the name of the element in question, a list of attribute definitions, and a closing `>`. Each attribute definition is a triplet that defines:

- The name of an attribute.
- The type of the attribute's value or an explicit set of possible values. Values defined explicitly by the DTD are case-insensitive. Please consult the section on basic HTML data types [p.49] for more information about attribute value types.
- Whether the default value of the attribute is implicit (keyword `"#IMPLIED"`), in which case the default value must be supplied by the user agent (in some cases via inheritance from parent elements); always required (keyword `"#REQUIRED"`), or fixed to the given value (keyword `"#FIXED"`). Some attribute definitions explicitly specify a default value for the attribute.

In this example, the `name` attribute is defined for the `MAP` element. The attribute is optional for this element.

```
<!ATTLIST MAP
  name CDATA #IMPLIED
  >
```

The type of values permitted for the attribute is given as `CDATA`, an SGML data type. `CDATA` is text that may contain character references [p.45].

For more information about "CDATA", "NAME", "ID", and other data types, please consult the section on HTML data types [p.49].

The following examples illustrate several attribute definitions:

```
rowspan NUMBER 1 -- number of rows spanned by cell --
http-equiv NAME #IMPLIED -- HTTP response header name --
id ID #IMPLIED -- document-wide unique id --
valign (top|middle|bottom|baseline) #IMPLIED
```

The rowspan attribute requires values of type NUMBER. The default value is given explicitly as "1". The optional http-equiv attribute requires values of type NAME. The optional id attribute requires values of type ID. The optional valign attribute is constrained to take values from the set {top, middle, bottom, baseline}.

DTD entities in attribute definitions

Attribute definitions may also contain parameter entity references.

In this example, we see that the attribute definition list for the LINK element begins with the "%attrs;" parameter entity.

```
<!ELEMENT LINK - O EMPTY -- a media-independent link --->
<!ATTLIST LINK
  %attrs;
  charset %Charset; #IMPLIED -- %coreattrs, %i18n, %events --
  href %URI; #IMPLIED -- char encoding of linked resource --
  hrefLang %LanguageCode; #IMPLIED -- language code --
  type %ContentType; #IMPLIED -- advisory content type --
  rel %LinkTypes; #IMPLIED -- forward link types --
  rev %LinkTypes; #IMPLIED -- reverse link types --
  media %MediaDesc; #IMPLIED -- for rendering on these media --
  >
```

Start tag: required, End tag: forbidden

The "%attrs;" parameter entity is defined as follows:

```
<!ENTITY % attrs "%coreattrs; %i18n; %events; ">
```

The "%coreattrs;" parameter entity in the "%attrs;" definition expands as follows:

```
<!ENTITY % coreattrs
  "id ID #IMPLIED -- document-wide unique id --
  class CDATA #IMPLIED -- space-separated list of classes --
  style %StyleSheet; #IMPLIED -- associated style info --
  title %Text; #IMPLIED -- advisory title ---"
  >
```

The "%attrs;" parameter entity has been defined for convenience since these attributes are defined for most HTML element types.

Similarly, the DTD defines the "%URI;" parameter entity as expanding into the string "CDATA".

```
<!ENTITY % URI "CDATA"
-- a Uniform Resource Identifier,
see [URI]
-->
```

As this example illustrates, the parameter entity "%URI;" provides readers of the DTD with more information as to the type of data expected for an attribute. Similar entities have been defined for "%Color", "%Charset", "%Length", "%Pixels", etc.

Boolean attributes

Some attributes play the role of boolean variables (e.g., the selected attribute for the OPTION element). Their appearance in the start tag of an element implies that the value of the attribute is "true". Their absence implies a value of "false".

Boolean attributes may legally take a single value: the name of the attribute itself (e.g., selected="selected").

This example defines the selected attribute to be a boolean attribute.

```
selected (selected) #IMPLIED -- option is pre-selected --
```

The attribute is set to "true" by appearing in the element's start tag:

```
<OPTION selected="selected" >
...contents...
</OPTION>
```

In HTML, boolean attributes may appear in *minimized form* -- the attribute's value appears alone in the element's start tag. Thus, selected may be set by writing:

```
<OPTION selected>
```

instead of:

```
<OPTION selected="selected">
```

Authors should be aware that many user agents **only** recognize the minimized form of boolean attributes and not the full form.

4 Conformance: requirements and recommendations

Contents

1. Definitions	37
2. SGML	39
3. The text/html content type	39

In this section, we begin the specification of HTML 4, starting with the contract between authors, documents, users, and user agents.

The key words "MUST", "MUST NOT", "REQUIRED", "SHALL", "SHALL NOT", "SHOULD", "SHOULD NOT", "RECOMMENDED", "MAY", and "OPTIONAL" in this document are to be interpreted as described in [RFC2119] [p.354]. However, for readability, these words do not appear in all uppercase letters in this specification.

At times, the authors of this specification recommend good practice for authors and user agents. These recommendations are not normative and conformance with this specification does not depend on their realization. These recommendations contain the expression "We recommend ...". This specification recommends "...", or some similar wording.

4.1 Definitions

HTML document

An HTML document is an SGML document that meets the constraints of this specification.

Author

An author is a person or program that writes or generates HTML documents. An *authoring tool* is a special case of an author, namely, it's a program that generates HTML.

We recommend that authors write documents that conform to the strict DTD [p.265] rather than the other DTDs defined by this specification. Please see the section on version information [p.60] for details about the DTDs defined in HTML 4.

User

A user is a person who interacts with a user agent to view, hear, or otherwise use a rendered HTML document.

HTML user agent

An HTML user agent is any device that interprets HTML documents. User agents include visual browsers (text-only and graphical), non-visual browsers (audio, Braille), search robots, proxies, etc.

A **conforming user agent** for HTML 4 is one that observes the mandatory conditions ("must") set forth in this specification, including the following points:

- A user agent should avoid imposing arbitrary length limits on attribute value literals (see the section on capacities in the SGML Declaration [p.263]). For introductory information on SGML attributes, please consult the section on attribute definitions [p.34].
- A user agent must ensure that rendering is unchanged by the presence or absence of start tags and end tags when the HTML DTD indicates that these are optional. See the section on element definitions [p.32] for introductory information on SGML elements.
- For reasons of backwards compatibility, we recommend that tools interpreting HTML 4 continue to support HTML 3.2 (see [HTML32] [p.356]) and HTML 2.0 (see [RFC1866] [p.356]).

Error conditions

This specification does not define how conforming user agents handle general error conditions, including how user agents behave when they encounter elements, attributes, attribute values, or entities not specified in this document.

However, for recommended error handling behavior, please consult the notes on invalid documents [p.334].

Deprecated

A deprecated element or attribute is one that has been outdated by newer constructs. Deprecated elements are defined in the reference manual in appropriate locations, but are clearly marked as deprecated. Deprecated elements may become obsolete in future versions of HTML.

User agents should continue to support deprecated elements for reasons of backward compatibility.

Definitions of elements and attributes clearly indicate which are deprecated.

This specification includes examples that illustrate how to avoid using deprecated elements. In most cases these depend on user agent support for style sheets. In general, authors should use style sheets to achieve stylistic and formatting effects rather than HTML presentational attributes. HTML presentational attributes have been deprecated when style sheet alternatives exist (see, for example, [CSS1] [p.353]).

Obsolete

An obsolete element or attribute is one for which there is no guarantee of support by a user agent. Obsolete elements are no longer defined in the specification, but are listed for historical purposes in the changes section [p.311] of the reference manual.

4.2 SGML

HTML 4 is an SGML application conforming to International Standard ISO 8879 -- *Standard Generalized Markup Language SGML* (defined in [ISO8879] [p.353]).

Examples in the text conform to the strict document type definition [p.265] unless the example in question refers to elements or attributes only defined by the transitional document type definition [p.279] or frameset document type definition [p.297]. For the sake of brevity, most of the examples in this specification do not begin with the document type declaration [p.60] that is mandatory at the beginning of each HTML document.

DTD fragments in element definitions come from the strict document type definition [p.265] except for the elements related to frames.

Please consult the section on HTML version information [p.60] for details about when to use the strict, transitional, or frameset DTD.

Comments appearing in the HTML 4 DTD [p.265] have no normative value; they are informative only.

User agents must not render SGML processing instructions (e.g., `<?full volume>`) or comments. For more information about this and other SGML features that may be legal in HTML but aren't widely supported by HTML user agents, please consult the section on SGML features with limited support. [p.337]

4.3 The text/html content type

HTML documents are sent over the Internet as a sequence of bytes accompanied by encoding information (described in the section on character encodings [p.42]). The structure of the transmission, termed a *message entity*, is defined by [RFC2045] [p.354] and [RFC2616] [p.354]. A message entity with a content type [p.53] of "text/html" represents an HTML document.

The content type for HTML documents is defined as follows:

Content type name:

text

Content subtype name:

html

Required parameters:

none

Optional parameters:

charset

Encoding considerations:

any encoding is allowed

Security considerations:

See the notes on security [p.350]

The optional parameter "charset" refers to the character encoding [p.42] used to represent the HTML document as a sequence of bytes. Legal values for this parameter are defined in the section on character encodings [p.42]. Although this parameter is optional, we recommend that it always be present.

5 HTML Document Representation

Contents

1. The Document Character Set	41
2. Character encodings	42
1. Choosing an encoding	42
• Notes on specific encodings	43
2. Specifying the character encoding	43
3. Character references	45
1. Numeric character references	45
2. Character entity references	46
4. Undisplayable characters	47

In this chapter, we discuss how HTML documents are represented on a computer and over the Internet.

The section on the document character set [p.41] addresses the issue of what abstract *characters* may be part of an HTML document. Characters include the Latin letter "A", the Cyrillic letter "I", the Chinese character meaning "water", etc.

The section on character encodings [p.42] addresses the issue of how those characters may be *represented* in a file or when transferred over the Internet. As some character encodings cannot directly represent all characters an author may want to include in a document, HTML offers other mechanisms, called character references [p.45], for referring to any character.

Since there are a great number of characters throughout human languages, and a great variety of ways to represent those characters, proper care must be taken so that documents may be understood by user agents around the world.

5.1 The Document Character Set

To promote interoperability, SGML requires that each application (including HTML) specify its *document character set*. A document character set consists of:

- A *Repertoire*: A set of abstract characters, such as the Latin letter "A", the Cyrillic letter "I", the Chinese character meaning "water", etc.
- *Code positions*: A set of integer references to characters in the repertoire.

Each SGML document (including each HTML document) is a sequence of characters from the repertoire. Computer systems identify each character by its code position; for example, in the ASCII character set, code positions 65, 66, and 67 refer to the characters 'A', 'B', and 'C', respectively.

The ASCII character set is not sufficient for a global information system such as the Web, so HTML uses the much more complete character set called the *Universal Character Set (UCS)*, defined in [ISO10646]. [p.353] This standard defines a

repertoire of thousands of characters used by communities all over the world.

The character set defined in [ISO10646] [p.353] is character-by-character equivalent to Unicode ([UNICODE] [p.355]). Both of these standards are updated from time to time with new characters, and the amendments should be consulted at the respective Web sites. In the current specification, "[ISO10646]" is used to refer to the document character set while "[UNICODE]" is reserved for references to the Unicode bidirectional text algorithm. [p.82]

The document character set, however, does not suffice to allow user agents to correctly interpret HTML documents as they are typically exchanged -- encoded as a sequence of bytes in a file or during a network transmission. User agents must also know the specific character encoding [p.42] that was used to transform the document character stream into a byte stream.

5.2 Character encodings

What this specification calls a *character encoding* is known by different names in other specifications (which may cause some confusion). However, the concept is largely the same across the Internet. Also, protocol headers, attributes, and parameters referring to character encodings share the same name -- "charset" -- and use the same values from the [IANA] [p.353] registry (see [CHARSETS] [p.355] for a complete list).

The "charset" parameter identifies a character encoding, which is a method of converting a sequence of bytes into a sequence of characters. This conversion fits naturally with the scheme of Web activity: servers send HTML documents to user agents as a stream of bytes; user agents interpret them as a sequence of characters. The conversion method can range from simple one-to-one correspondence to complex switching schemes or algorithms.

A simple one-byte-per-character encoding technique is not sufficient for text strings over a character repertoire as large as [ISO10646] [p.353]. There are several different encodings of parts of [ISO10646] [p.353] in addition to encodings of the entire character set (such as UCS-4).

5.2.1 Choosing an encoding

Authoring tools (e.g., text editors) may encode HTML documents in the character encoding of their choice, and the choice largely depends on the conventions used by the system software. These tools may employ any convenient encoding that covers most of the characters contained in the document, provided the encoding is correctly labeled. [p.43] Occasional characters that fall outside this encoding may still be represented by character references [p.45]. These always refer to the document character set, not the character encoding.

Servers and proxies may change a character encoding (called *transcoding*) on the fly to meet the requests of user agents (see section 14.2 of [RFC2616] [p.354], the "Accept-Charset" HTTP request header). Servers and proxies do not have to serve a

document in a character encoding that covers the entire document character set.

Commonly used character encodings on the Web include ISO-8859-1 (also referred to as "Latin-1", usable for most Western European languages), ISO-8859-5 (which supports Cyrillic), SHIFT_JIS (a Japanese encoding), EUC-JP (another Japanese encoding), and UTF-8 (an encoding of ISO 10646 using a different number of bytes for different characters). Names for character encodings are case-insensitive, so that for example "SHIFT_JIS", "Shift_JIS", and "shift_jis" are equivalent.

This specification does not mandate which character encodings a user agent must support.

Conforming user agents [p.38] must correctly map to ISO 10646 all characters in any character encodings that they recognize (or they must behave as if they did).

Notes on specific encodings

When HTML text is transmitted in UTF-16 (charset=UTF-16), text data should be transmitted in network byte order ("big-endian", high-order byte first) in accordance with [ISO10646] [p.353], Section 6.3 and [UNICODE] [p.355], clause C3, page 3-1.

Furthermore, to maximize chances of proper interpretation, it is recommended that documents transmitted as UTF-16 always begin with a ZERO-WIDTH NON-BREAKING SPACE character (hexadecimal FEFF, also called Byte Order Mark (BOM)) which, when byte-reversed, becomes hexadecimal FFFE, a character guaranteed never to be assigned. Thus, a user-agent receiving a hexadecimal FFFE as the first bytes of a text would know that bytes have to be reversed for the remainder of the text.

The UTF-1 transformation format of [ISO10646] [p.353] (registered by IANA as ISO-10646-UTF-1), should not be used. For information about ISO 8859-8 and the bidirectional algorithm, please consult the section on bidirectionality and character encoding [p.86].

5.2.2 Specifying the character encoding

How does a server determine which character encoding applies for a document it serves? Some servers examine the first few bytes of the document, or check against a database of known files and encodings. Many modern servers give Web masters more control over charset configuration than old servers do. Web masters should use these mechanisms to send out a "charset" parameter whenever possible, but should take care not to identify a document with the wrong "charset" parameter value.

How does a user agent know which character encoding has been used? The server should provide this information. The most straightforward way for a server to inform the user agent about the character encoding of the document is to use the "charset" parameter of the "Content-Type" header field of the HTTP protocol ([RFC2616] [p.354], sections 3.4 and 14.17) For example, the following HTTP header announces that the character encoding is EUC-JP:

```
Content-Type: text/html; charset=EUC-JP
```

Please consult the section on conformance [p.37] for the definition of text/html [p.39].

The HTTP protocol ([RFC2616] [p.354], section 3.7.1) mentions ISO-8859-1 as a default character encoding when the "charset" parameter is absent from the "Content-Type" header field. In practice, this recommendation has proved useless because some servers don't allow a "charset" parameter to be sent, and others may not be configured to send the parameter. Therefore, user agents must not assume any default value for the "charset" parameter.

To address server or configuration limitations, HTML documents may include explicit information about the document's character encoding; the META element can be used to provide user agents with this information.

For example, to specify that the character encoding of the current document is "EUC-JP", a document should include the following META declaration:

```
<META http-equiv="Content-Type" content="text/html; charset=EUC-JP">
```

The META declaration must only be used when the character encoding is organized such that ASCII-valued bytes stand for ASCII characters (at least until the META element is parsed). META declarations should appear as early as possible in the HEAD element.

For cases where neither the HTTP protocol nor the META element provides information about the character encoding of a document, HTML also provides the charset attribute on several elements. By combining these mechanisms, an author can greatly improve the chances that, when the user retrieves a resource, the user agent will recognize the character encoding.

To sum up, conforming user agents must observe the following priorities when determining a document's character encoding (from highest priority to lowest):

1. An HTTP "charset" parameter in a "Content-Type" field.
2. A META declaration with "http-equiv" set to "Content-Type" and a value set for "charset".
3. The charset attribute set on an element that designates an external resource.

In addition to this list of priorities, the user agent may use heuristics and user settings. For example, many user agents use a heuristic to distinguish the various encodings used for Japanese text. Also, user agents typically have a user-definable, local default character encoding which they apply in the absence of other indicators.

User agents may provide a mechanism that allows users to override incorrect "charset" information. However, if a user agent offers such a mechanism, it should only offer it for browsing and not for editing, to avoid the creation of Web pages marked with an incorrect "charset" parameter.

Note. *If, for a specific application, it becomes necessary to refer to characters outside [ISO10646] [p.353] , characters should be assigned to a private zone to avoid conflicts with present or future versions of the standard. This is highly discouraged, however, for reasons of portability.*

5.3 Character references

A given character encoding may not be able to express all characters of the document character set. For such encodings, or when hardware or software configurations do not allow users to input some document characters directly, authors may use SGML character references. Character references are a character encoding-independent mechanism for entering any character from the document character set.

Character references in HTML may appear in two forms:

- Numeric character references (either decimal or hexadecimal).
- Character entity references.

Character references within comments have no special meaning; they are comment data only.

Note. *HTML provides other ways to present character data, in particular inline images [p.459] .*

Note. *In SGML, it is possible to eliminate the final ";" after a character reference in some cases (e.g., at a line break or immediately before a tag). In other circumstances it may not be eliminated (e.g., in the middle of a word). We strongly suggest using the ";" in all cases to avoid problems with user agents that require this character to be present.*

5.3.1 Numeric character references

Numeric character references specify the code position [p.41] of a character in the document character set. Numeric character references may take two forms:

- The syntax "&#D;", where *D* is a decimal number, refers to the ISO 10646 decimal character number *D*.
- The syntax "&#xH;" or "&#XH;", where *H* is a hexadecimal number, refers to the ISO 10646 hexadecimal character number *H*. Hexadecimal numbers in numeric character references are case-insensitive.

Here are some examples of numeric character references:

- å (in decimal) represents the letter "a" with a small circle above it (used, for example, in Norwegian).
- å (in hexadecimal) represents the same character.
- å (in hexadecimal) represents the same character as well.
- И (in decimal) represents the Cyrillic capital letter "I".

- 水 (in hexadecimal) represents the Chinese character for water.

Note. *Although the hexadecimal representation is not defined in [ISO8879] [p.353] , it is expected to be in the revision, as described in [WEBSGML] [p.355] . This convention is particularly useful since character standards generally use hexadecimal representations.*

5.3.2 Character entity references

In order to give authors a more intuitive way of referring to characters in the document character set, HTML offers a set of *character entity references*. Character entity references use symbolic names so that authors need not remember code positions. [p.41] For example, the character entity reference å refers to the lowercase "a" character topped with a ring; "å" is easier to remember than å.

HTML 4 does not define a character entity reference for every character in the document character set. For instance, there is no character entity reference for the Cyrillic capital letter "I". Please consult the full list of character references [p.299] defined in HTML 4.

Character entity references are case-sensitive. Thus, Å refers to a different character (uppercase A, ring) than å (lowercase a, ring).

Four character entity references deserve special mention since they are frequently used to escape special characters:

- "<" represents the < sign.
- ">" represents the > sign.
- "&" represents the & sign.
- """ represents the " mark.

Authors wishing to put the "<" character in text should use "<"; (ASCII decimal 60) to avoid possible confusion with the beginning of a tag (start tag open delimiter). Similarly, authors should use ">"; (ASCII decimal 62) in text instead of ">" to avoid problems with older user agents that incorrectly perceive this as the end of a tag (tag close delimiter) when it appears in quoted attribute values.

Authors should use "&" (ASCII decimal 38) instead of "&" to avoid confusion with the beginning of a character reference (entity reference open delimiter). Authors should also use "&" in attribute values since character references are allowed within CDATA [p.50] attribute values.

Some authors use the character entity reference """ to encode instances of the double quote mark (") since that character may be used to delimit attribute values.

5.4 Undisplayable characters

A user agent may not be able to render all characters in a document meaningfully, for instance, because the user agent lacks a suitable font, a character has a value that may not be expressed in the user agent's internal character encoding, etc.

Because there are many different things that may be done in such cases, this document does not prescribe any specific behavior. Depending on the implementation, undisplayable characters may also be handled by the underlying display system and not the application itself. In the absence of more sophisticated behavior, for example tailored to the needs of a particular script or language, we recommend the following behavior for user agents:

1. Adopt a clearly visible, but unobtrusive mechanism to alert the user of missing resources.
2. If missing characters are presented using their numeric representation, use the hexadecimal (not decimal) form since this is the form used in character set standards.

6 Basic HTML data types

Contents

1. Case information	49
2. SGML basic types	50
3. Text strings	50
4. URIs	51
5. Colors	51
1. Notes on using colors	52
6. Lengths	52
7. Content types (MIME types)	53
8. Language codes	53
9. Character encodings	53
10. Single characters	53
11. Dates and times	54
12. Link types	54
13. Media descriptors	56
14. Script data	57
15. Style sheet data	57
16. Frame target names	57

This section of the specification describes the basic data types that may appear as an element's content or an attribute's value.

For introductory information about reading the HTML DTD, please consult the SGML tutorial [p.27].

6.1 Case information

Each attribute definition includes information about the case-sensitivity of its values. The case information is presented with the following keys:

- CS** The value is case-sensitive (i.e., user agents interpret "a" and "A" differently).
- CI** The value is case-insensitive (i.e., user agents interpret "a" and "A" as the same).
- CN** The value is not subject to case changes, e.g., because it is a number or a character from the document character set.
- CA** The element or attribute definition itself gives case information.
- CT** Consult the type definition for details about case-sensitivity.

If an attribute value is a list, the keys apply to every value in the list, unless otherwise indicated.

6.2 SGML basic types

The document type definition [p.265] specifies the syntax of HTML element content and attribute values using SGML tokens (e.g., PCDATA, CDATA, NAME, ID, etc.). See [ISO8879] [p.353] for their full definitions. The following is a summary of key information:

- **CDATA** is a sequence of characters from the document character set and may include character entities. User agents should interpret attribute values as follows:
 - Replace character entities with characters,
 - Ignore line feeds,
 - Replace each carriage return or tab with a single space.

User agents may ignore leading and trailing white space in CDATA attribute values (e.g., " myval " may be interpreted as "myval"). Authors should not declare attribute values with leading or trailing white space.

For some HTML 4 attributes with CDATA attribute values, the specification imposes further constraints on the set of legal values for the attribute that may not be expressed by the DTD.

Although the `STYLE` and `SCRIPT` elements use CDATA for their data model, for these elements, CDATA must be handled differently by user agents. Markup and entities must be treated as raw text and passed to the application as is. The first occurrence of the character sequence "<" (end-tag open delimiter) is treated as terminating the end of the element's content. In valid documents, this would be the end tag for the element.

- **ID** and **NAME** tokens must begin with a letter ([A-Za-z]) and may be followed by any number of letters, digits ([0-9]), hyphens ("-"), underscores ("_"), colons (":"), and periods (".").
- **IDREF** and **IDREFS** are references to ID tokens defined by other attributes. IDREF is a single token and IDREFS is a space-separated list of tokens.
- **NUMBER** tokens must contain at least one digit ([0-9]).

6.3 Text strings

A number of attributes (%Text; [p.266] in the DTD) take text that is meant to be "human readable". For introductory information about attributes, please consult the tutorial discussion of attributes [p.29].

6.4 URIs

This specification uses the term URI as defined in [URI] [p.355] (see also [RFC1630] [p.356]).

Note that URIs include URLs (as defined in [RFC1738] [p.354] and [RFC1808] [p.354]).

Relative URIs are resolved to full URIs using a base URI. [RFC1808] [p.354], section 3, defines the normative algorithm for this process. For more information about base URIs, please consult the section on base URIs [p.157] in the chapter on links [p.145].

URIs are represented in the DTD by the parameter entity %URI; [p.266].

















URIs in general are case-sensitive. [p.49] There may be URIs, or parts of URIs, where case doesn't matter (e.g., machine names), but identifying these may not be easy. Users should always consider that URIs are case-sensitive (to be on the safe side).

Please consult the appendix for information about non-ASCII characters in URI attribute values [p.334].

6.5 Colors

The attribute value type "color" (%Color; [p.281]) refers to color definitions as specified in [SRGB] [p.354]. A color value may either be a hexadecimal number (prefixed by a hash mark) or one of the following sixteen color names. The color names are case-insensitive. [p.49]

Color names and sRGB values

	Black = "#000000"		Green = "#008000"
	Silver = "#C0C0C0"		Lime = "#00FF00"
	Gray = "#808080"		Olive = "#808000"
	White = "#FFFFFF"		Yellow = "#FFFF00"
	Maroon = "#800000"		Navy = "#000080"
	Red = "#FF0000"		Blue = "#0000FF"
	Purple = "#800080"		Teal = "#008080"
	Fuchsia = "#FF00FF"		Aqua = "#00FFFF"

Thus, the color values "#800080" and "Purple" both refer to the color purple.

6.5.1 Notes on using colors

Although colors can add significant amounts of information to documents and make them more readable, please consider the following guidelines when including color in your documents:

- The use of HTML elements and attributes for specifying color is deprecated [p.38]. You are encouraged to use style sheets [p.183] instead.
- Don't use color combinations that cause problems for people with color blindness in its various forms.
- If you use a background image or set the background color, then be sure to set the various text colors as well.
- Colors specified with the BODY and FONT elements and bgcolor on tables look different on different platforms (e.g., workstations, Macs, Windows, and LCD panels vs. CRTs), so you shouldn't rely entirely on a specific effect. In the future, support for the [SRGB] [p.354] color model together with ICC color profiles should mitigate this problem.
- When practical, adopt common conventions to minimize user confusion.

6.6 Lengths

HTML specifies three types of length values for attributes:

1. **Pixels.** The value (%Pixels; [p.270] in the DTD) is an integer that represents the number of pixels of the canvas (screen, paper). Thus, the value "50" means fifty pixels. For normative information about the definition of a pixel, please consult [CSS1] [p.353].
2. **Length.** The value (%Length; [p.270] in the DTD) may be either a %Pixel; or a percentage of the available horizontal or vertical space. Thus, the value "50%" means half of the available space.
3. **MultiLength.** The value (%MultiLength; [p.270] in the DTD) may be a %Length; or a *relative length*. A relative length has the form "i*", where "i" is an integer. When allotting space among elements competing for that space, user agents allot pixel and percentage lengths first, then divide up remaining available space among relative lengths. Each relative length receives a portion of the available space that is proportional to the integer preceding the "*". The value "i*" is equivalent to "1*i". Thus, if 60 pixels of space are available after the user agent allots pixel and percentage space, and the competing relative lengths are 1*, 2*, and 3*, the 1* will be allotted 10 pixels, the 2* will be allotted 20 pixels, and the 3* will be allotted 30 pixels.

Length values are case-neutral. [p.49]

6.7 Content types (MIME types)

Note. A "media type" (defined in [RFC2045] [p.354] and [RFC2046] [p.354]) specifies the nature of a linked resource. This specification employs the term "content type" rather than "media type" in accordance with current usage. Furthermore, in this specification, "media type" may refer to the media [p.56] where a user agent renders a document.

This type is represented in the DTD by %ContentType. [p.266]

Content types are case-insensitive. [p.49]

Examples of content types include "text/html", "image/png", "image/gif", "video/mpeg", "text/css", and "audio/basic". For the current list of registered MIME types, please consult [MIMETYPES]. [p.354]

6.8 Language codes

The value of attributes whose type is a language code (%LanguageCode [p.266] in the DTD) refers to a language code as specified by [RFC1766] [p.354], section 2. For information on specifying language codes in HTML, please consult the section on language codes [p.80]. Whitespace is not allowed within the language-code.

Language codes are case-insensitive. [p.49]

6.9 Character encodings

The "charset" attributes (%Charset [p.266] in the DTD) refer to a character encoding as described in the section on character encodings [p.42]. Values must be strings (e.g., "euc-jp") from the IANA registry (see [CHARSETS] [p.355] for a complete list).

Names of character encodings are case-insensitive. [p.49]

User agents must follow the steps set out in the section on specifying character encodings [p.43] in order to determine the character encoding of an external resource.

6.10 Single characters

Certain attributes call for a single character from the document character set [p.41]. These attributes take the %Character [p.266] type in the DTD.

Single characters may be specified with character references [p.45] (e.g., "&").

6.11 Dates and times

[ISO8601] [p.353] allows many options and variations in the representation of dates and times. The current specification uses one of the formats described in the profile [DATE TIME] [p.353] for its definition of legal date/time strings (%DateTime [p.266] in the DTD).

The format is:

```
YYYY-MM-DDThh:mm:ssTZD
```

where:

```

YYYY = four-digit year
MM   = two-digit month (01=January, etc.)
DD   = two-digit day of month (01 through 31)
hh   = two digits of hour (00 through 23) (am/pm NOT allowed)
mm   = two digits of minute (00 through 59)
ss   = two digits of second (00 through 59)
TZD  = time zone designator

```

The time zone designator is one of:

Z

indicates UTC (Coordinated Universal Time). The "Z" must be uppercase.

+hh:mm

indicates that the time is a local time which is hh hours and mm minutes ahead of UTC.

-hh:mm

indicates that the time is a local time which is hh hours and mm minutes behind UTC.

Exactly the components shown here must be present, with exactly this punctuation. Note that the "T" appears literally in the string (it must be uppercase), to indicate the beginning of the time element, as specified in [ISO8601] [p.353]

If a generating application does not know the time to the second, it may use the value "00" for the seconds (and minutes and hours if necessary).

Note. [DATE TIME] [p.353] does not address the issue of leap seconds.

6.12 Link types

Authors may use the following recognized link types, listed here with their conventional interpretations. In the DTD, %LinkTypes [p.266] refers to a space-separated list of link types. White space characters are not permitted within link types.

These link types are case-insensitive, [p.49] i.e., "Alternate" has the same meaning as "alternate".

User agents, search engines, etc. may interpret these link types in a variety of ways. For example, user agents may provide access to linked documents through a navigation bar.

Alternate

Designates substitute versions for the document in which the link occurs. When used together with the `lang` attribute, it implies a translated version of the document. When used together with the `media` attribute, it implies a version designed for a different medium (or media).

Stylesheet

Refers to an external style sheet. See the section on external style sheets [p.190] for details. This is used together with the link type "Alternate" for user-selectable alternate style sheets.

Start

Refers to the first document in a collection of documents. This link type tells search engines which document is considered by the author to be the starting point of the collection.

Next

Refers to the next document in a linear sequence of documents. User agents may choose to preload the "next" document, to reduce the perceived load time.

Prev

Refers to the previous document in an ordered series of documents. Some user agents also support the synonym "Previous".

Contents

Refers to a document serving as a table of contents. Some user agents also support the synonym `ToC` (from "Table of Contents").

Index

Refers to a document providing an index for the current document.

Glossary

Refers to a document providing a glossary of terms that pertain to the current document.

Copyright

Refers to a copyright statement for the current document.

Chapter

Refers to a document serving as a chapter in a collection of documents.

Section

Refers to a document serving as a section in a collection of documents.

Subsection

Refers to a document serving as a subsection in a collection of documents.

Appendix

Refers to a document serving as an appendix in a collection of documents.

Help

Refers to a document offering help (more information, links to other sources information, etc.)

Bookmark

Refers to a bookmark. A bookmark is a link to a key entry point within an extended document. The `title` attribute may be used, for example, to label the

bookmark. Note that several bookmarks may be defined in each document.

Authors may wish to define additional link types not described in this specification. If they do so, they should use a profile [p.68] to cite the conventions used to define the link types. Please see the `profile` attribute of the `HEAD` element for more details.

For further discussions about link types, please consult the section on links in HTML documents [p.145].

6.13 Media descriptors

The following is a list of recognized media descriptors (%MediaDesc [p.266] in the DTD).

screen

Intended for non-paged computer screens.

tty

Intended for media using a fixed-pitch character grid, such as teletypes, terminals, or portable devices with limited display capabilities.

tv

Intended for television-type devices (low resolution, color, limited scrollability).

projection

Intended for projectors.

handheld

Intended for handheld devices (small screen, monochrome, bitmapped graphics, limited bandwidth).

print

Intended for paged, opaque material and for documents viewed on screen in print preview mode.

braille

Intended for braille tactile feedback devices.

aural

Intended for speech synthesizers.

all

Suitable for all devices.

Future versions of HTML may introduce new values and may allow parameterized values. To facilitate the introduction of these extensions, conforming user agents must be able to parse the `media` attribute value as follows:

1. The value is a comma-separated list of entries. For example,

```
media="screen, 3d-glasses, print and resolution > 90dpi"
```

is mapped to:

```
"screen"
"3d-glasses"
"print and resolution > 90dpi"
```

- Each entry is truncated just before the first character that isn't a US ASCII letter [a-zA-Z] (ISO 10646 hex 41-5a, 61-7a), digit [0-9] (hex 30-39), or hyphen (hex 2d). In the example, this gives:

```
"screen"
"3d-glasses"
"print"
```

- A case-sensitive [p.49] match is then made with the set of media types defined above. User agents may ignore entries that don't match. In the example we are left with `screen` and `print`.

Note. *Style sheets may include media-dependent variations within them (e.g., the CSS @media construct). In such cases it may be appropriate to use "media=all".*

6.14 Script data

Script data (`%Script`; [p.266] in the DTD [p.265]) can be the content of the `SCRIPT` element and the value of intrinsic event attributes [p.251] . User agents must not evaluate script data as HTML markup but instead must pass it on as data to a script engine.

The case-sensitivity of script data depends on the scripting language.

Please note that script data that is element content may not contain character references [p.45] , but script data that is the value of an attribute may contain them. The appendix provides further information about specifying non-HTML data [p.336] .

6.15 Style sheet data

Style sheet data (`%StyleSheet`; [p.266] in the DTD [p.265]) can be the content of the `STYLE` element and the value of the `style` attribute. User agents must not evaluate style data as HTML markup.

The case-sensitivity of style data depends on the style sheet language.

Please note that style sheet data that is element content may not contain character references [p.45] , but style sheet data that is the value of an attribute may contain them. The appendix provides further information about specifying non-HTML data [p.336] .

6.16 Frame target names

Except for the reserved names listed below, frame target names (`%FrameTarget`; [p.281] in the DTD) must begin with an alphabetic character (a-zA-Z). User agents should ignore all other target names.

The following target names are reserved and have special meanings.

_blank

The user agent should load the designated document in a new, unnamed window.

_self

The user agent should load the document in the same frame as the element that refers to this target.

_parent

The user agent should load the document into the immediate `FRAMESET` parent of the current frame. This value is equivalent to `_self` if the current frame has no parent.

_top

The user agent should load the document into the full, original window (thus canceling all other frames). This value is equivalent to `_self` if the current frame has no parent.

7 The global structure of an HTML document

Contents

1. Introduction to the structure of an HTML document	59
2. HTML version information	60
3. The HTML element	61
4. The document head	62
1. The HEAD element	62
2. The TITLE element	62
3. The title attribute	63
4. Meta data	64
• Specifying meta data	64
• The META element	65
• Meta data profiles	68
5. The document body	69
1. The BODY element	69
2. Element identifiers: the id and class attributes	71
3. Block-level and inline elements	73
4. Grouping elements: the DIV and SPAN elements	73
5. Headings: The H1, H2, H3, H4, H5, H6 elements	75
6. The ADDRESS element	76

7.1 Introduction to the structure of an HTML document

An HTML 4 document is composed of three parts:

1. a line containing HTML version information [p.60] ,
2. a declarative header section (delimited by the HEAD element),
3. a body, which contains the document's actual content. The body may be implemented by the BODY element or the FRAMESET element.

White space (spaces, newlines, tabs, and comments) may appear before or after each section. Sections 2 and 3 should be delimited by the HTML element.

Here's an example of a simple HTML document:

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01//EN"
"http://www.w3.org/TR/html4/strict.dtd">
<HTML>
  <HEAD>
    <TITLE>My first HTML document</TITLE>
  </HEAD>
  <BODY>
    <P>Hello world!
  </BODY>
</HTML>
```

7.2 HTML version information

A valid HTML document declares what version of HTML is used in the document. The *document type declaration* names the document type definition (DTD) in use for the document (see [ISO8879] [p.353]).

HTML 4.01 specifies three DTDs, so authors must include one of the following document type declarations in their documents. The DTDs vary in the elements they support.

- The HTML 4.01 Strict DTD [p.265] includes all elements and attributes that have not been deprecated [p.38] or do not appear in frameset documents. For documents that use this DTD, use this document type declaration:

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01//EN"
"http://www.w3.org/TR/html4/strict.dtd">
```

- The HTML 4.01 Transitional DTD [p.279] includes everything in the strict DTD plus deprecated elements and attributes (most of which concern visual presentation). For documents that use this DTD, use this document type declaration:

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN"
"http://www.w3.org/TR/html4/loose.dtd">
```

- The HTML 4.01 Frameset DTD [p.297] includes everything in the transitional DTD plus frames as well. For documents that use this DTD, use this document type declaration:

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01 Frameset//EN"
"http://www.w3.org/TR/html4/frameset.dtd">
```

The URI in each document type declaration allows user agents to download the DTD and any entity sets [p.299] that are needed. The following (relative) URIs refer to DTDs and entity sets for HTML 4:

- "strict.dtd" -- default strict DTD
- "loose.dtd" -- loose DTD
- "frameset.dtd" -- DTD for frameset documents
- "HTMLat1.ent" -- Latin-1 entities
- "HTMLsymbol.ent" -- Symbol entities
- "HTMLspecial.ent" -- Special entities

The binding between public identifiers and files can be specified using a catalog file following the format recommended by the Oasis Open Consortium (see [OASISOPEN] [p.356]). A sample catalog file for HTML 4.01 [p.262] is included at the beginning of the section on SGML reference information for HTML. The last two letters of the declaration indicate the language of the DTD. For HTML, this is always English ("EN").

Note. As of the 24 December version of HTML 4.01, the HTML Working Group commits to the following policy:

- Any changes to future HTML 4 DTDs will not invalidate documents that conform to the DTDs of the present specification. The HTML Working Group reserves the right to correct known bugs.
- Software conforming to the DTDs of the present specification may ignore features of future HTML 4 DTDs that it does not recognize.

This means that in a document type declaration, authors may safely use a system identifier that refers to the latest version of an HTML 4 DTD. Authors may also choose to use a system identifier that refers to a specific (dated) version of an HTML 4 DTD when validation to that particular DTD is required. W3C will make every effort to make archival documents indefinitely available at their original address in their original form.

7.3 The HTML element

```
<!ENTITY % html.content "HEAD, BODY">
<!ELEMENT HTML O O (%html.content;) -- document root element -->
<!ATTLIST HTML
  %i18n;
  >
  -- lang, dir --
```

Start tag: optional, End tag: optional

Attributes defined elsewhere

version = cdata [p.50] [CN] [p.49]

Deprecated. [p.38] The value of this attribute specifies which HTML DTD version governs the current document. This attribute has been deprecated because it is redundant with version information [p.60] provided by the document type declaration.

Attributes defined elsewhere

- lang (language information [p.79]), dir (text direction [p.82])

After document type declaration, the remainder of an HTML document is contained by the HTML element. Thus, a typical HTML document has this structure:

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01//EN"
"http://www.w3.org/TR/html4/strict.dtd">
<HTML>
...The head, body, etc. goes here...
</HTML>
```

7.4 The document head

7.4.1 The HEAD element

```
<!-- %head.misc; defined earlier on as "SCRIPT|STYLE|META|LINK|OBJECT" -->
<!ENTITY % head.content "TITLE & BASE?>
<!ELEMENT HEAD O O (%head.content;) +(%head.misc;) -- document head -->
<!ATTLIST HEAD
  %i18n;
  profile %URI; #IMPLIED -- named dictionary of meta info --
  >
  -- lang, dir --
```

Start tag: optional, End tag: optional

Attribute definitions

profile = uri [p.51] [CT] [p.49]

This attribute specifies the location of one or more meta data profiles, separated by white space. For future extensions, user agents should consider the value to be a list even though this specification only considers the first URI to be significant. Profiles [p.68] are discussed below in the section on meta data [p.64]

Attributes defined elsewhere

- lang (language information [p.79]), dir (text direction [p.82])

The HEAD element contains information about the current document, such as its title, keywords that may be useful to search engines, and other data that is not considered document content. User agents do not generally render elements that appear in the HEAD as content. They may, however, make information in the HEAD available to users through other mechanisms.

7.4.2 The TITLE element

```
<!-- The TITLE element is not considered part of the flow of text.
It should be displayed, for example as the page header or
window title. Exactly one title is required per document.
-->
```

```
<!ELEMENT TITLE - - (#PCDATA) -(%head.misc;) -- document title -->
<!ATTLIST TITLE %i18n;
```

Start tag: required, End tag: required

Attributes defined elsewhere

- lang (language information [p.79]), dir (text direction [p.82])

Every HTML document **must** have a `TITLE` element in the `HEAD` section.

Authors should use the `TITLE` element to identify the contents of a document. Since users often consult documents out of context, authors should provide context-rich titles. Thus, instead of a title such as "Introduction", which doesn't provide much contextual background, authors should supply a title such as "Introduction to Medieval Bee-Keeping" instead.

For reasons of accessibility, user agents must always make the content of the `TITLE` element available to users (including `TITLE` elements that occur in frames). The mechanism for doing so depends on the user agent (e.g., as a caption, spoken). Titles may contain character entities [p.299] (for accented characters, special characters, etc.), but may not contain other markup (including comments). Here is a sample document title:

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01//EN"
"http://www.w3.org/TR/html4/strict.dtd">
<HTML>
<HEAD>
<TITLE>A study of population dynamics</TITLE>
... other head elements...
</HEAD>
<BODY>
... document body...
</BODY>
</HTML>
```

7.4.3 The `title` attribute

Attribute definitions

`title = text` [p.50] [CS] [p.49]

This attribute offers advisory information about the element for which it is set.

Unlike the `TITLE` element, which provides information about an entire document and may only appear once, the `title` attribute may annotate any number of elements. Please consult an element's definition to verify that it supports this attribute.

Values of the `title` attribute may be rendered by user agents in a variety of ways. For instance, visual browsers frequently display the title as a "tool tip" (a short message that appears when the pointing device pauses over an object). Audio user agents may speak the title information in a similar context. For example, setting the attribute on a link allows user agents (visual and non-visual) to tell users about the nature of the linked resource:

```
...some text...
Here's a photo of
<A href="http://someplace.com/heatstuff.gif" title="me scuba diving">
me scuba diving last summer
</A>
...some more text...
```

The `title` attribute has an additional role when used with the `LINK` element to designate an external style sheet. [p.190] Please consult the section on links and style sheets [p.155] for details.

Note. To improve the quality of speech synthesis for cases handled poorly by standard techniques, future versions of HTML may include an attribute for encoding phonemic and prosodic information.

7.4.4 Meta data

Note. The W3C Resource Description Framework (see [RDF-10] [p.356]) became a W3C Recommendation in February 1999. RDF allows authors to specify machine-readable metadata about HTML documents and other network-accessible resources.

HTML lets authors specify meta data -- information about a document rather than document content -- in a variety of ways.

For example, to specify the author of a document, one may use the `META` element as follows:

```
<META name="Author" content="Dave Raggett">
```

The `META` element specifies a property (here "Author") and assigns a value to it (here "Dave Raggett").

This specification does not define a set of legal meta data properties. The meaning of a property and the set of legal values for that property should be defined in a reference lexicon called a profile [p.68]. For example, a profile designed to help search engines index documents might define properties such as "author", "copyright", "keywords", etc.

Specifying meta data

In general, specifying meta data involves two steps:

1. Declaring a property and a value for that property. This may be done in two ways:
 1. From within a document, via the `META` element.
 2. From outside a document, by linking to meta data via the `LINK` element (see the section on link types [p.54]).
2. Referring to a profile [p.68] where the property and its legal values are defined. To designate a profile, use the `profile` attribute of the `HEAD` element.

Note that since a profile is defined for the `HEAD` element, the same profile applies to all `META` and `LINK` elements in the document head.

User agents are not required to support meta data mechanisms. For those that choose to support meta data, this specification does not define how meta data should be interpreted.

The META element

```
<ELEMENT META - O EMPTY
<!ATTLIST META
  %!l18n;
  http-equiv NAME #IMPLIED -- lang, dir, for use with content --
  name NAME #IMPLIED -- HTTP response header name --
  content CDATA #REQUIRED -- metainformation name --
  scheme CDATA #IMPLIED -- associated information --
  >
  -- generic metainformation -->
  -- lang, dir, for use with content --
  -- HTTP response header name --
  -- metainformation name --
  -- associated information --
  -- select form of content --
```

Start tag: required, End tag: forbidden

Attribute definitions

For the following attributes, the permitted values and their interpretation are profile dependent:

`name = name [p.50] [CS] [p.49]`

This attribute identifies a property name. This specification does not list legal values for this attribute.

`content = cdata [p.50] [CS] [p.49]`

This attribute specifies a property's value. This specification does not list legal values for this attribute.

`scheme = cdata [p.50] [CS] [p.49]`

This attribute names a scheme to be used to interpret the property's value (see the section on profiles [p.68] for details).

`http-equiv = name [p.50] [CI] [p.49]`

This attribute may be used in place of the `name` attribute. HTTP servers use this attribute to gather information for HTTP response message headers.

Attributes defined elsewhere

- `lang` (language information [p.79]), `dir` (text direction [p.82])

The `META` element can be used to identify properties of a document (e.g., author, expiration date, a list of key words, etc.) and assign values to those properties. This specification does not define a normative set of properties.

Each `META` element specifies a property/value pair. The `name` attribute identifies the property and the `content` attribute specifies the property's value.

For example, the following declaration sets a value for the `Author` property:

```
<META name="Author" content="Dave Raggett">
```

The `lang` attribute can be used with `META` to specify the language for the value of the `content` attribute. This enables speech synthesizers to apply language dependent pronunciation rules.

In this example, the author's name is declared to be French:

```
<META name="Author" lang="fr" content="Arnaud Le Hors">
```

Note. The `META` element is a generic mechanism for specifying meta data. However, some HTML elements and attributes already handle certain pieces of meta data and may be used by authors instead of `META` to specify those pieces: the `TITLE` element, the `ADDRESS` element, the `INS` and `DEL` elements, the `title` attribute, and the `cite` attribute.

Note. When a property specified by a `META` element takes a value that is a URI [p.51], some authors prefer to specify the meta data via the `LINK` element. Thus, the following meta data declaration:

```
<META name="DC.identifier"
content="http://www.ietf.org/rfc/rfc1866.txt">
```

might also be written:

```
<LINK rel="DC.identifier"
type="text/plain"
href="http://www.ietf.org/rfc/rfc1866.txt">
```

META and HTTP headers

The `http-equiv` attribute can be used in place of the `name` attribute and has a special significance when documents are retrieved via the Hypertext Transfer Protocol (HTTP). HTTP servers may use the property name specified by the `http-equiv` attribute to create an [RFC822] [p.356] -style header in the HTTP response. Please see the HTTP specification ([RFC2616] [p.354]) for details on valid HTTP headers.

The following sample `META` declaration:

```
<META http-equiv="Expires" content="Tue, 20 Aug 1996 14:25:27 GMT">
```

will result in the HTTP header:

```
Expires: Tue, 20 Aug 1996 14:25:27 GMT
```

This can be used by caches to determine when to fetch a fresh copy of the associated document.

Note. Some user agents support the use of `META` to refresh the current page after a specified number of seconds, with the option of replacing it by a different URI. Authors should **not** use this technique to forward users to different pages, as this makes the page inaccessible to some users. Instead, automatic page forwarding should be done using server-side redirects.

META and search engines

A common use for `META` is to specify keywords that a search engine may use to improve the quality of search results. When several `META` elements provide language-dependent information about a document, search engines may filter on the `lang` attribute to display search results using the language preferences of the user. For example,

```

<-- For speakers of US English -->
<META name="keywords" lang="en-us"
  content="vacation, Greece, sunshine">
<-- For speakers of British English -->
<META name="keywords" lang="en"
  content="holiday, Greece, sunshine">
<-- For speakers of French -->
<META name="keywords" lang="fr"
  content="vacances, Grèce, soleil">

```

The effectiveness of search engines can also be increased by using the `LINK` element to specify links to translations of the document in other languages, links to versions of the document in other media (e.g., PDF), and, when the document is part of a collection, links to an appropriate starting point for browsing the collection.

Further help is provided in the section on helping search engines index your Web site [p.339].

META and PICS

The Platform for Internet Content Selection (PICS, specified in [PICS] [p.356]) is an infrastructure for associating labels (meta data) with Internet content. Originally designed to help parents and teachers control what children can access on the Internet, it also facilitates other uses for labels, including code signing, privacy, and intellectual property rights management.

This example illustrates how one can use a `META` declaration to include a PICS 1.1 label:

```

<HEAD>
<META http-equiv="PICS-Label" content='
(PICS-1.1 "http://www.gcf.org/v2.5"
  labels on "1994.11.05T08:15-0500"
  until "1995.12.31T23:59-0000"
  for "http://w3.org/PICS/Overview.html"
  ratings (suds 0.5 density 0 color/hue 1)
'
</HEAD>

```

META and default information

The `META` element may be used to specify the default information for a document in the following instances:

- The default scripting language [p.253].
- The default style sheet language [p.186].
- The document character encoding [p.41].

The following example specifies the character encoding [p.41] for a document as being ISO-8859-5

```

<META http-equiv="Content-Type" content="text/html; charset=ISO-8859-5">

```

Meta data profiles

The profile attribute of the `HEAD` specifies the location of a meta data profile. The value of the profile attribute is a URI. User agents may use this URI in two ways:

- As a globally unique name. User agents may be able to recognize the name (without actually retrieving the profile) and perform some activity based on known conventions for that profile. For instance, search engines could provide an interface for searching through catalogs of HTML documents, where these documents all use the same profile for representing catalog entries.
- As a link. User agents may dereference the URI and perform some activity based on the actual definitions within the profile (e.g., authorize the usage of the profile within the current HTML document). This specification does not define formats for profiles.

This example refers to a hypothetical profile that defines useful properties for document indexing. The properties defined by this profile -- including "author", "copyright", "keywords", and "date" -- have their values set by subsequent `META` declarations.

```

<HEAD profile="http://www.acme.com/profiles/core">
<TITLE>How to complete Memorandum cover sheets</TITLE>
<META name="author" content="John Doe">
<META name="copyright" content="&copy; 1997 Acme Corp.">
<META name="keywords" content="corporate, guidelines, cataloging">
<META name="date" content="1994-11-06T08:49:37+00:00">
</HEAD>

```

As this specification is being written, it is common practice to use the date formats described in [RFC2616] [p.354], section 3.3. As these formats are relatively hard to process, we recommend that authors use the [ISO8601] [p.353] date format. For more information, see the sections on the *INS* and *DEL* elements.

The scheme attribute allows authors to provide user agents more context for the correct interpretation of meta data. At times, such additional information may be critical, as when meta data may be specified in different formats. For example, an author might specify a date in the (ambiguous) format "10-9-97"; does this mean 9 October 1997 or 10 September 1997? The scheme attribute value "Month-Day-Year" would disambiguate this date value.

At other times, the scheme attribute may provide helpful but non-critical information to user agents.

For example, the following scheme declaration may help a user agent determine that the value of the "identifier" property is an ISBN code number:

```

<META scheme="ISBN" name="identifier" content="0-8230-2355-9">

```

Values for the scheme attribute depend on the property name and the associated profile.

Note. *One sample profile is the Dublin Core (see [DCORE] [p.355]). This profile defines a set of recommended properties for electronic bibliographic descriptions, and is intended to promote interoperability among disparate description models.*

7.5 The document body

7.5.1 The BODY element

```
<!ELEMENT BODY O (%block;|SCRIPT)+ +(INS|DEL) -- document body -->
<!ATTLIST BODY
  %attrs;
  onLoad %Script; #IMPLIED -- the document has been loaded --
  onUnload %Script; #IMPLIED -- the document has been removed --
>
```

Start tag: optional. End tag: optional

Attribute definitions

`background = uri` [p.51] [CT] [p.49]

Deprecated. [p.38] The value of this attribute is a URI that designates an image resource. The image generally tiles the background (for visual browsers).

`text = color` [p.51] [CI] [p.49]

Deprecated. [p.38] This attribute sets the foreground color for text (for visual browsers).

`link = color` [p.51] [CI] [p.49]

Deprecated. [p.38] This attribute sets the color of text marking unvisited

hypertext links (for visual browsers).

`vlink = color` [p.51] [CI] [p.49]

Deprecated. [p.38] This attribute sets the color of text marking visited hypertext links (for visual browsers).

`alink = color` [p.51] [CI] [p.49]

Deprecated. [p.38] This attribute sets the color of text marking hypertext links when selected by the user (for visual browsers).

Attributes defined elsewhere

- `id, class` (document-wide identifiers [p.71])
- `lang` (language information [p.79]), `dir` (text direction [p.82])
- `title` (element title [p.63])
- `style` (inline style information [p.186])
- `bgcolor` (background color [p.195])
- `onLoad, onUnload` (intrinsic events [p.254])
- `onClick, onDoubleClick, onMouseDown, onMouseUp, onMouseOver, onMouseMove, onMouseOut, onKeyPress, onKeyDown, onKeyUp` (intrinsic events [p.254])

The body of a document contains the document's content. The content may be presented by a user agent in a variety of ways. For example, for visual browsers, you can think of the body as a canvas where the content appears: text, images, colors, graphics, etc. For audio user agents, the same content may be spoken. Since style sheets [p.183] are now the preferred way to specify a document's presentation, the presentational attributes of BODY have been deprecated [p.38].

DEPRECATED EXAMPLE:

The following HTML fragment illustrates the use of the deprecated [p.38] attributes. It sets the background color of the canvas to white, the text foreground color to black, and the color of hyperlinks to red initially, fuchsia when activated, and maroon once visited.

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN"
  "http://www.w3.org/TR/html4/loose.dtd">
<HTML>
<HEAD>
  <TITLE>A study of population dynamics</TITLE>
</HEAD>
<BODY bgcolor="white" text="black"
  link="red" alink="fuchsia" vlink="maroon">
  ... document body...
</BODY>
</HTML>
```

Using style sheets [p.183], the same effect could be accomplished as follows:

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01//EN"
  "http://www.w3.org/TR/html4/strict.dtd">
<HTML>
<HEAD>
  <TITLE>A study of population dynamics</TITLE>
  <STYLE type="text/css">
  BODY { background: white; color: black }
  A:link { color: red }
  A:visited { color: maroon }
  A:active { color: fuchsia }
</STYLE>
</HEAD>
<BODY>
  ... document body...
</BODY>
</HTML>
```

Using external (linked) style sheets gives you the flexibility to change the presentation without revising the source HTML document:

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01//EN"
  "http://www.w3.org/TR/html4/strict.dtd">
<HTML>
<HEAD>
<TITLE>A study of population dynamics</TITLE>
<LINK rel="stylesheet" type="text/css" href="smartstyle.css">
</HEAD>
<BODY>
  ... document body...
</BODY>
</HTML>
```

Framesets and HTML bodies. Documents that contain framesets replace the BODY element by the FRAMESET element. Please consult the section on frames [p.205] for more information.

7.5.2 Element identifiers: the id and class attributes

Attribute definitions

`id = name` [p.50] [CS] [p.49]

This attribute assigns a name to an element. This name must be unique in a document.

`class = cdata-ist` [p.50] [CS] [p.49]

This attribute assigns a class name or set of class names to an element. Any number of elements may be assigned the same class name or names. Multiple class names must be separated by white space characters.

The `id` attribute assigns a unique identifier to an element (which may be verified by an SGML parser). For example, the following paragraphs are distinguished by their `id` values:

```
<P id="myparagraph"> This is a uniquely named paragraph.</P>
<P id="yourparagraph"> This is also a uniquely named paragraph.</P>
```

The `id` attribute has several roles in HTML:

- As a style sheet [p.183] selector.
- As a target anchor [p.145] for hypertext links.
- As a means to reference a particular element from a script [p.254].
- As the name of a declared OBJECT element.
- For general purpose processing by user agents (e.g. for identifying fields when extracting data from HTML pages into a database, translating HTML documents into other formats, etc.).

The `class` attribute, on the other hand, assigns one or more class names to an element; the element may be said to belong to these classes. A class name may be shared by several element instances. The `class` attribute has several roles in HTML:

- As a style sheet [p.183] selector (when an author wishes to assign style information to a set of elements).
- For general purpose processing by user agents.

In the following example, the SPAN element is used in conjunction with the `id` and `class` attributes to markup document messages. Messages appear in both English and French versions.

```
<!-- English messages -->
<P><SPAN id="msg1" class="info" lang="en">Variable declared twice</SPAN>
<P><SPAN id="msg2" class="warning" lang="en">Undeclared variable</SPAN>
<P><SPAN id="msg3" class="error" lang="en">Bad syntax for variable name</SPAN>
<!-- French messages -->
<P><SPAN id="msg1" class="info" lang="fr">Variable d&eacute;clar&eacute;e deux fois</SPAN>
<P><SPAN id="msg2" class="warning" lang="fr">Variable ind&eacute;clar&eacute;e</SPAN>
<P><SPAN id="msg3" class="error" lang="fr">Erreur de syntaxe pour variable</SPAN>
```

The following CSS style rules would tell visual user agents to display informational messages in green, warning messages in yellow, and error messages in red:

```
SPAN.info { color: green }
SPAN.warning { color: yellow }
SPAN.error { color: red }
```

Note that the French "msg1" and the English "msg1" may not appear in the same document since they share the same `id` value. Authors may make further use of the `id` attribute to refine the presentation of individual messages, make them target anchors, etc.

Almost every HTML element may be assigned identifier and class information.

Suppose, for example, that we are writing a document about a programming language. The document is to include a number of preformatted examples. We use the PRE element to format the examples. We also assign a background color (green) to all instances of the PRE element belonging to the class "example".

```
<HEAD>
<TITLE>... document title ...</TITLE>
<STYLE type="text/css">
PRE.example { background : green }
</STYLE>
</HEAD>
<BODY>
<PRE class="example" id="example-1">
...example code here...
</PRE>
</BODY>
```

By setting the `id` attribute for this example, we can (1) create a hyperlink to it and (2) override class style information with instance style information.

Note. The `id` attribute shares the same name space as the name attribute when used for anchor names. Please consult the section on anchors with `id` [p.152] for more information.

7.5.3 Block-level and inline elements

Certain HTML elements that may appear in `BODY` are said to be "block-level" while others are "inline" (also known as "text level"). The distinction is founded on several notions:

Content model

Generally, block-level elements may contain inline elements and other block-level elements. Generally, inline elements may contain only data and other inline elements. Inherent in this structural distinction is the idea that block elements create "larger" structures than inline elements.

Formatting

By default, block-level elements are formatted differently than inline elements. Generally, block-level elements begin on new lines, inline elements do not. For information about white space, line breaks, and block formatting, please consult the section on text [p.89].

Directionality

For technical reasons involving the [UNICODE] [p.355] bidirectional text algorithm, block-level and inline elements differ in how they inherit directionality information. For details, see the section on inheritance of text direction [p.83].

Style sheets [p.183] provide the means to specify the rendering of arbitrary elements, including whether an element is rendered as block or inline. In some cases, such as an inline style for list elements, this may be appropriate, but generally speaking, authors are discouraged from overriding the conventional interpretation of HTML elements in this way.

The alteration of the traditional presentation idioms for block level and inline elements also has an impact on the bidirectional text algorithm. See the section on the effect of style sheets on bidirectionality [p.88] for more information.

7.5.4 Grouping elements: the DIV and SPAN elements

```
<ELEMENT DIV - - (%Flow?)* -- generic language/style container -->
<|ATTLIST DIV
  %attrs;
  >
<ELEMENT SPAN - - (%inline?)* -- generic language/style container -->
<|ATTLIST SPAN
  %attrs;
  >
```

Start tag: required, End tag: required

Attributes defined elsewhere

- `id`, `class` (document-wide identifiers [p.71])
- `lang` (language information [p.79]), `dir` (text direction [p.82])
- `title` (element title [p.63])
- `style` (inline style information [p.186])

- `align` (alignment [p.195])
- `onclick`, `ondblclick`, `onmousedown`, `onmouseup`, `onmouseover`, `onmousemove`, `onmouseout`, `onkeypress`, `onkeydown`, `onkeyup` (intrinsic events [p.254])

The `DIV` and `SPAN` elements, in conjunction with the `id` and `class` attributes, offer a generic mechanism for adding structure to documents. These elements define content to be inline (`SPAN`) or block-level (`DIV`) but impose no other presentational idioms on the content. Thus, authors may use these elements in conjunction with style sheets [p.183], the `lang` attribute, etc., to tailor HTML to their own needs and tastes.

Suppose, for example, that we wanted to generate an HTML document based on a database of client information. Since HTML does not include elements that identify objects such as "client", "telephone number", "email address", etc., we use `DIV` and `SPAN` to achieve the desired structural and presentational effects. We might use the `TABLE` element as follows to structure the information:

```
<!-- Example of data from the client database: -->
<!-- Name: Stephane Boyera, Tel: (212) 555-1212, Email: sb@foo.org -->
<DIV id="client-boyera" class="client">
<P><SPAN class="client-title">Client information:</SPAN>
<TABLE class="client-data">
<TR><TH>Last name:<TD>Boyera</TR>
<TR><TH>First name:<TD>Stephane</TR>
<TR><TH>Tel:<TD>(212) 555-1212</TR>
<TR><TH>Email:<TD>sb@foo.org</TR>
</TABLE>
</DIV>

<DIV id="client-lafon" class="client">
<P><SPAN class="client-title">Client information:</SPAN>
<TABLE class="client-data">
<TR><TH>Last name:<TD>Lafon</TR>
<TR><TH>First name:<TD>Yves</TR>
<TR><TH>Tel:<TD>(617) 555-1212</TR>
<TR><TH>Email:<TD>yves@coucou.com</TR>
</TABLE>
</DIV>
```

Later, we may easily add style sheet declarations to fine tune the presentation of these database entries.

For another example of usage, please consult the example in the section on the `class` and `id` attributes [p.72].

Visual user agents generally place a line break before and after `DIV` elements, for instance:

```
<P>aaaaaaaa<DIV>bbbbbbbb</DIV><DIV>cccccc<P>cccccc</DIV>
```

which is typically rendered as:

```
aaaaaaaa
bbbbbbbb
cccccc
cccccc
```

7.5.5 Headings: The H1, H2, H3, H4, H5, H6 elements

```
<!ENTITY % heading "H1|H2|H3|H4|H5|H6">
<!--
There are six levels of headings from H1 (the most important)
to H6 (the least important).
-->
<!ELEMENT {%heading;} - - (%inline;)* -- heading -->
<!ATTLIST {%heading;}
  %attrs;
  -- %coreattrs, %i18n, %events --
>
```

Start tag: required, End tag: required

Attributes defined elsewhere

- id, class (document-wide identifiers [p.71])
- lang (language information [p.79]), dir (text direction [p.82])
- title (element title [p.63])
- style (inline style information [p.186])
- align (alignment [p.195])
- onclick, ondblclick, onmousedown, onmouseup, onmouseover, onmousemove, onmouseout, onkeypress, onkeydown, onkeyup (intrinsic events [p.254])

A heading element briefly describes the topic of the section it introduces. Heading information may be used by user agents, for example, to construct a table of contents for a document automatically.

There are six levels of headings in HTML with H1 as the most important and H6 as the least. Visual browsers usually render more important headings in larger fonts than less important ones.

The following example shows how to use the DIV element to associate a heading with the document section that follows it. Doing so allows you to define a style for the section (color the background, set the font, etc.) with style sheets.

```
<DIV class="section" id="forest-elephants" >
<H1>Forest elephants</H1>
<P>In this section, we discuss the lesser known forest elephants.
...this section continues...
<DIV class="subsection" id="forest-habitat" >
<H2>Habitat</H2>
<P>Forest elephants do not live in trees but among them.
...this subsection continues...
</DIV>
</DIV>
```

This structure may be decorated with style information such as:

```
<HEAD>
<TITLE>... document title ...</TITLE>
<STYLE type="text/css">
DIV.section { text-align: justify; font-size: 12pt }
DIV.subsection { text-indent: 2em }
H1 { font-style: italic; color: green }
H2 { color: green }
</STYLE>
</HEAD>
```

Numbered sections and references

HTML does not itself cause section numbers to be generated from headings. This facility may be offered by user agents, however. Soon, style sheet languages such as CSS will allow authors to control the generation of section numbers (handy for forward references in printed documents, as in "See section 7.2").

Some people consider skipping heading levels to be bad practice. They accept H1 H2 H1 while they do not accept H1 H3 H1 since the heading level H2 is skipped.

7.5.6 The ADDRESS element

```
<!ELEMENT ADDRESS - - (%inline;)* -- information on author -->
<!ATTLIST ADDRESS
  %attrs;
  -- %coreattrs, %i18n, %events --
>
```

Start tag: required, End tag: required

Attributes defined elsewhere

- id, class (document-wide identifiers [p.71])
- lang (language information [p.79]), dir (text direction [p.82])
- title (element title [p.63])
- style (inline style information [p.186])
- onclick, ondblclick, onmousedown, onmouseup, onmouseover, onmousemove, onmouseout, onkeypress, onkeydown, onkeyup (intrinsic events [p.254])

The ADDRESS element may be used by authors to supply contact information for a document or a major part of a document such as a form. This element often appears at the beginning or end of a document.

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><BR>
$Date: 1999/12/24 23:07:14 $
</ADDRESS>
```

8 Language information and text direction

Contents

1. Specifying the language of content: the <code>Lang</code> attribute	79
1. Language codes	80
2. Inheritance of language codes	81
3. Interpretation of language codes	81
2. Specifying the direction of text and tables: the <code>dir</code> attribute	82
1. Introduction to the bidirectional algorithm	82
2. Inheritance of text direction information	83
3. Setting the direction of embedded text	84
4. Overriding the bidirectional algorithm: the <code>BDO</code> element	85
5. Character references for directionality and joining control	87
6. The effect of style sheets on bidirectionality	88

This section of the document discusses two important issues that affect the internationalization of HTML: specifying the language (the `Lang` attribute) and direction (the `dir` attribute) of text in a document.

8.1 Specifying the language of content: the `Lang` attribute

Attribute definitions

`Lang` = *language-code* [p.53] [CI] [p.49]

This attribute specifies the base language of an element's attribute values and text content. The default value of this attribute is unknown.

Language information specified via the `Lang` attribute may be used by a user agent to control rendering in a variety of ways. Some situations where author-supplied language information may be helpful include:

- Assisting search engines
- Assisting speech synthesizers
- Helping a user agent select glyph variants for high quality typography
- Helping a user agent choose a set of quotation marks
- Helping a user agent make decisions about hyphenation [p.96], ligatures, and spacing
- Assisting spell checkers and grammar checkers

The `Lang` attribute specifies the language of element content and attribute values; whether it is relevant for a given attribute depends on the syntax and semantics of the attribute and the operation involved.

The intent of the `Lang` attribute is to allow user agents to render content more meaningfully based on accepted cultural practice for a given language. This does not imply that user agents should render characters that are atypical for a particular language in less meaningful ways; user agents must make a best attempt to render all characters, regardless of the value specified by `Lang`.

For instance, if characters from the Greek alphabet appear in the midst of English text:

```
<P><Q Lang="en">Her super-powers were the result of  
&gamma;-radiation.</Q> he explained.</P>
```

a user agent (1) should try to render the English content in an appropriate manner (e.g., in its handling the quotation marks) and (2) must make a best attempt to render `γ` even though it is not an English character.

Please consult the section on undisplayable characters [p.47] for related information.

8.1.1 Language codes

The `Lang` attribute's value is a language code that identifies a natural language spoken, written, or otherwise used for the communication of information among people. Computer languages are explicitly excluded from language codes.

[RFC1766] [p.354] defines and explains the language codes that must be used in HTML documents.

Briefly, language codes consist of a primary code and a possibly empty series of subcodes:

```
language-code = primary-code ( "-" subcode ) *
```

Here are some sample language codes:

- "en": English
- "en-US": the U.S. version of English.
- "en-cockney": the Cockney version of English.
- "i-navajo": the Navajo language spoken by some Native Americans.
- "x-klingson": The primary tag "x" indicates an experimental language tag

Two-letter primary codes are reserved for [ISO639] [p.353] language abbreviations. Two-letter codes include fr (French), de (German), it (Italian), nl (Dutch), el (Greek), es (Spanish), pt (Portuguese), ar (Arabic), he (Hebrew), ru (Russian), zh (Chinese), ja (Japanese), hi (Hindi), ur (Urdu), and sa (Sanskrit).

Any two-letter subcode is understood to be a [ISO3166] [p.353] country code.

8.1.2 Inheritance of language codes

An element inherits language code information according to the following order of precedence (highest to lowest):

- The `lang` attribute set for the element itself.
- The closest parent element that has the `lang` attribute set (i.e., the `lang` attribute is inherited).
- The HTTP "Content-Language" header (which may be configured in a server). For example:


```
Content-Language: en-cockney
```
- User agent default values and user preferences.

In this example, the primary language of the document is French ("fr"). One paragraph is declared to be in Spanish ("es"), after which the primary language returns to French. The following paragraph includes an embedded Japanese ("ja") phrase, after which the primary language returns to French.

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01//EN"
"http://www.w3.org/TR/html4/strict.dtd">
<HTML lang="fr">
<HEAD>
<TITLE>Un document multilingue</TITLE>
</HEAD>
<BODY>
...Interpreted as French...
<p lang="es">...Interpreted as Spanish...
<p>...Interpreted as French again...
<p>...French text interrupted by<EM lang="ja">some
Japanese</EM>French begins here again...
</BODY>
</HTML>
```

Note. Table cells may inherit `lang` values not from its parent but from the first cell in a span. Please consult the section on alignment inheritance [p.133] for details.

8.1.3 Interpretation of language codes

In the context of HTML, a language code should be interpreted by user agents as a hierarchy of tokens rather than a single token. When a user agent adjusts rendering according to language information (say, by comparing style sheet language codes and `lang` values), it should always favor an exact match, but should also consider matching primary codes to be sufficient. Thus, if the `lang` attribute value of "en-US" is set for the HTML element, a user agent should prefer style information that matches "en-US" first, then the more general value "en".

Note. Language code hierarchies do not guarantee that all languages with a common prefix will be understood by those fluent in one or more of those languages. They do allow a user to request this commonality when it is true for that user.

8.2 Specifying the direction of text and tables: the `dir` attribute

Attribute definitions

`dir = LTR | RTL | C1` [p.49]

This attribute specifies the base direction of directionally neutral text (i.e., text that doesn't have inherent directionality as defined in [UNICODE] [p.355]) in an element's content and attribute values. It also specifies the directionality of tables [p.115]. Possible values:

- LTR: Left-to-right text or table.
- RTL: Right-to-left text or table.

In addition to specifying the language of a document with the `lang` attribute, authors may need to specify the base directionality (left-to-right or right-to-left) of portions of a document's text, of table structure, etc. This is done with the `dir` attribute.

The [UNICODE] [p.355] specification assigns directionality to characters and defines a (complex) algorithm for determining the proper directionality of text. If a document does not contain a displayable right-to-left character, a conforming user agent is not required to apply the [UNICODE] [p.355] bidirectional algorithm. If a document contains right-to-left characters, and if the user agent displays these characters, the user agent must use the bidirectional algorithm.

Although Unicode specifies special characters that deal with text direction, HTML offers higher-level markup constructs that do the same thing: the `dir` attribute (do not confuse with the `DIR` element) and the `BDO` element. Thus, to express a Hebrew quotation, it is more intuitive to write

```
<Q lang="he" dir="rtl">...a Hebrew quotation...</Q>
```

than the equivalent with Unicode references:

```
&#x202B;&#x05F4;...a Hebrew quotation...&#x05F4;&#x202C;
```

User agents must **not** use the `lang` attribute to determine text directionality.

The `dir` attribute is inherited and may be overridden. Please consult the section on the inheritance of text direction information [p.83] for details.

8.2.1 Introduction to the bidirectional algorithm

The following example illustrates the expected behavior of the bidirectional algorithm. It involves English, a left-to-right script, and Hebrew, a right-to-left script.

Consider the following example text:

```
english1 HEBREW2 english3 HEBREW4 english5 HEBREW6
```

The characters in this example (and in all related examples) are stored in the computer the way they are displayed here: the first character in the file is "e", the second is "n", and the last is "g".

Suppose the predominant language of the document containing this paragraph is English. This means that the base direction is left-to-right. The correct presentation of this line would be:

```
english1 2WERBEH english3 4WERBEH english5 6WERBEH
<----->
  H           H           H           <----->
-----
  E
```

The dotted lines indicate the structure of the sentence: English predominates and some Hebrew text is embedded. Achieving the correct presentation requires no additional markup since the Hebrew fragments are reversed correctly by user agents applying the bidirectional algorithm.

If, on the other hand, the predominant language of the document is Hebrew, the base direction is right-to-left. The correct presentation is therefore:

```
6WERBEH english5 4WERBEH english3 2WERBEH english1
----->
  E           E           E           <----->
-----
  H
```

In this case, the whole sentence has been presented as right-to-left and the embedded English sequences have been properly reversed by the bidirectional algorithm.

8.2.2 Inheritance of text direction information

The Unicode bidirectional algorithm requires a base text direction for text blocks. To specify the base direction of a block-level element, set the element's `dir` attribute. The default value of the `dir` attribute is "ltr" (left-to-right text).

When the `dir` attribute is set for a block-level element, it remains in effect for the duration of the element and any nested block-level elements. Setting the `dir` attribute on a nested element overrides the inherited value.

To set the base text direction for an entire document, set the `dir` attribute on the `HTML` element.

For example:

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01//EN"
"http://www.w3.org/TR/html4/strict.dtd">
<HTML dir="RTL">
<HEAD>
<TITLE>...a right-to-left title...</TITLE>
</HEAD>
<BODY>
...right-to-left text...
<P dir="ltr">...left-to-right text...</P>
<P>...right-to-left text again...</P>
</HTML>
```

Inline elements, on the other hand, do not inherit the `dir` attribute. This means that an inline element without a `dir` attribute does **not** open an additional level of embedding with respect to the bidirectional algorithm. (Here, an element is considered to be block-level or inline based on its default presentation. Note that the `INS` and `DEL` elements can be block-level or inline depending on their context.)

8.2.3 Setting the direction of embedded text

The [UNICODE] [p.355] bidirectional algorithm automatically reverses embedded character sequences according to their inherent directionality (as illustrated by the previous examples). However, in general only one level of embedding can be accounted for. To achieve additional levels of embedded direction changes, you must make use of the `dir` attribute on an inline element.

Consider the same example text as before:

```
english1 HEBREW2 english3 HEBREW4 english5 HEBREW6
```

Suppose the predominant language of the document containing this paragraph is English. Furthermore, the above English sentence contains a Hebrew section extending from HEBREW2 through HEBREW4 and the Hebrew section contains an English quotation (english3). The desired presentation of the text is thus:

```
english1 4WERBEH english3 2WERBEH english5 6WERBEH
----->
  E           <----->
-----
  H
-----
  E
```

To achieve two embedded direction changes, we must supply additional information, which we do by delimiting the second embedding explicitly. In this example, we use the `SPAN` element and the `dir` attribute to mark up the text:

```
english1 <SPAN dir="RTL">HEBREW2 english3 HEBREW4</SPAN> english5 HEBREW6
```

Authors may also use special Unicode characters to achieve multiple embedded direction changes. To achieve left-to-right embedding, surround embedded text with the characters LEFT-TO-RIGHT EMBEDDING ("LRE", hexadecimal 202A) and POP DIRECTIONAL FORMATTING ("PDF", hexadecimal 202C). To achieve right-to-left embedding, surround embedded text with the characters RIGHT-TO-LEFT

EMBEDDING ("RTE", hexadecimal 202B) and PDF.

Using HTML directionality markup with Unicode characters. Authors and designers of authoring software should be aware that conflicts can arise if the `dir` attribute is used on inline elements (including BDO) concurrently with the corresponding [UNICODE] [p.355] formatting characters. Preferably one or the other should be used exclusively. The markup method offers a better guarantee of document structural integrity and alleviates some problems when editing bidirectional HTML text with a simple text editor, but some software may be more apt at using the [UNICODE] [p.355] characters. If both methods are used, great care should be exercised to insure proper nesting of markup and directional embedding or override, otherwise, rendering results are undefined.

8.2.4 Overriding the bidirectional algorithm: the BDO element

```
<ELEMENT BDO - - (%inline;)* -- I18N Bidi over-ride -->
<!ATTLIST BDO
  %coreattrs; -- id, class, style, title --
  lang %LanguageCode; #IMPLIED -- language code --
  dir (ltr|rtl) #REQUIRED -- directionality --
  >
```

Start tag: required, End tag: required

Attribute definitions

`dir = LTR | RTL | C` [p.49]

This mandatory attribute specifies the base direction of the element's text content. This direction overrides the inherent directionality of characters as defined in [UNICODE] [p.355]. Possible values:

- LTR: Left-to-right text.
- RTL: Right-to-left text.

Attributes defined elsewhere

- lang (language information [p.79])

The bidirectional algorithm and the `dir` attribute generally suffice to manage embedded direction changes. However, some situations may arise when the bidirectional algorithm results in incorrect presentation. The BDO element allows authors to turn off the bidirectional algorithm for selected fragments of text.

Consider a document containing the same text as before:

```
english1 HEBREW2 english3 HEBREW4 english5 HEBREW6
```

but assume that this text has already been put in visual order. One reason for this may be that the MIME standard ([RFC2045] [p.354], [RFC1556] [p.354]) favors visual order, i.e., that right-to-left character sequences are inserted right-to-left in the byte stream. In an email, the above might be formatted, including line breaks, as:

```
english1 2WERBEH english3
4WERBEH english5 6WERBEH
```

This conflicts with the [UNICODE] [p.355] bidirectional algorithm, because that algorithm would invert 2WERBEH, 4WERBEH, and 6WERBEH a second time, displaying the Hebrew words left-to-right instead of right-to-left.

The solution in this case is to override the bidirectional algorithm by putting the Email excerpt in a PRE element (to conserve line breaks) and each line in a BDO element, whose `dir` attribute is set to LTR:

```
<PRE>
<BDO dir="LTR">english1 2WERBEH english3</BDO>
<BDO dir="LTR">4WERBEH english5 6WERBEH</BDO>
</PRE>
```

This tells the bidirectional algorithm "Leave me left-to-right!" and would produce the desired presentation:

```
english1 2WERBEH english3
4WERBEH english5 6WERBEH
```

The BDO element should be used in scenarios where absolute control over sequence order is required (e.g., multi-language part numbers). The `dir` attribute is mandatory for this element.

Authors may also use special Unicode characters to override the bidirectional algorithm -- LEFT-TO-RIGHT OVERRIDE (202D) or RIGHT-TO-LEFT OVERRIDE (hexadecimal 202E). The POP DIRECTIONAL FORMATTING (hexadecimal 202C) character ends either bidirectional override.

Note. Recall that conflicts can arise if the `dir` attribute is used on inline elements (including BDO) concurrently with the corresponding [UNICODE] [p.355] formatting characters.

Bidirectionality and character encoding According to [RFC1555] [p.354] and [RFC1556] [p.354], there are special conventions for the use of "charset" parameter values to indicate bidirectional treatment in MIME mail, in particular to distinguish between visual, implicit, and explicit directionality. The parameter value "ISO-8859-8" (for Hebrew) denotes visual encoding, "ISO-8859-8-i" denotes implicit bidirectionality, and "ISO-8859-8-e" denotes explicit directionality.

Because HTML uses the Unicode bidirectionality algorithm, conforming documents encoded using ISO 8859-8 must be labeled as "ISO-8859-8-i". Explicit directional control is also possible with HTML, but cannot be expressed with ISO 8859-8, so "ISO-8859-8-e" should not be used.

The value "ISO-8859-8" implies that the document is formatted visually, misusing some markup (such as TABLE with right alignment and no line wrapping) to ensure reasonable display on older user agents that do not handle bidirectionality. Such documents do not conform to the present specification. If necessary, they can be made to conform to the current specification (and at the same time will be displayed correctly on older user agents) by adding BDO markup where necessary. Contrary to

what is said in [RFC1555] [p.354] and [RFC1556] [p.354] , ISO-8859-6 (Arabic) is *not* visual ordering.

8.2.5 Character references for directionality and joining control

Since ambiguities sometimes arise as to the directionality of certain characters (e.g., punctuation), the [UNICODE] [p.355] specification includes characters to enable their proper resolution. Also, Unicode includes some characters to control joining behavior where this is necessary (e.g., some situations with Arabic letters). HTML 4 includes character references [p.299] for these characters.

The following DTD excerpt presents some of the directional entities:

```
<!ENTITY zwnj CDATA "&#204;"--zero width non-joiner-->
<!ENTITY zwj CDATA "&#205;"--zero width joiner-->
<!ENTITY lrm CDATA "&#206;"--left-to-right mark-->
<!ENTITY rlm CDATA "&#207;"--right-to-left mark-->
```

The `zwnj` entity is used to block joining behavior in contexts where joining will occur but shouldn't. The `zwj` entity does the opposite; it forces joining when it wouldn't occur but should. For example, the Arabic letter "HEH" is used to abbreviate "Hijri", the name of the Islamic calendar system. Since the isolated form of "HEH" looks like the digit five as employed in Arabic script (based on Indic digits), in order to prevent confusing "HEH" as a final digit five in a year, the initial form of "HEH" is used. However, there is no following context (i.e., a joining letter) to which the "HEH" can join. The `zwj` character provides that context.

Similarly, in Persian texts, there are cases where a letter that normally would join a subsequent letter in a cursive connection should not. The character `zwnj` is used to block joining in such cases.

The other characters, `lrm` and `rlm`, are used to force directionality of directionally neutral characters. For example, if a double quotation mark comes between an Arabic (right-to-left) and a Latin (left-to-right) letter, the direction of the quotation mark is not clear (is it quoting the Arabic text or the Latin text?). The `lrm` and `rlm` characters have a directional property but no width and no word/line break property. Please consult [UNICODE] [p.355] for more details.

Mirrored character glyphs. In general, the bidirectional algorithm does not mirror character glyphs but leaves them unaffected. An exception are characters such as parentheses (see [UNICODE] [p.355] , table 4-7). In cases where mirroring is desired, for example for Egyptian Hieroglyphs, Greek Boustrophedon, or special design effects, this should be controlled with styles.

8.2.6 The effect of style sheets on bidirectionality

In general, using style sheets to change an element's visual rendering from block-level to inline or vice-versa is straightforward. However, because the bidirectional algorithm relies on the inline/block-level distinction [p.83] , special care must be taken during the transformation.

When an inline element that does not have a `dir` attribute is transformed to the style of a block-level element by a style sheet, it inherits the `dir` attribute from its closest parent block element to define the base direction of the block.

When a block element that does not have a `dir` attribute is transformed to the style of an inline element by a style sheet, the resulting presentation should be equivalent, in terms of bidirectional formatting, to the formatting obtained by explicitly adding a `dir` attribute (assigned the inherited value) to the transformed element.

9 Text

Contents

1. White space	89
2. Structured text	90
1. Phrase elements: EM, STRONG, DFN, CODE, SAMP, KBD, VAR, CITE, ABBR, and ACRONYM	90
2. Quotations: The BLOCKQUOTE and Q elements	92
• Rendering quotations	93
3. Subscripts and superscripts: the SUB and SUP elements	94
3. Lines and Paragraphs	94
1. Paragraphs: the P element	95
2. Controlling line breaks	95
• Forcing a line break: the BR element	96
• Prohibiting a line break	96
3. Hyphenation	96
4. Preformatted text: The PRE element	97
5. Visual rendering of paragraphs	98
4. Marking document changes: The INS and DEL elements	99

The following sections discuss issues surrounding the structuring of text. Elements that present text [p. 195] (alignment elements, font elements, style sheets, etc.) are discussed elsewhere in the specification. For information about characters, please consult the section on the document character set. [p.41]

9.1 White space

The document character set [p.41] includes a wide variety of white space characters. Many of these are typographic elements used in some applications to produce particular visual spacing effects. In HTML, only the following characters are defined as *white space characters*:

- ASCII space ()
- ASCII tab ()
- ASCII form feed ()
- Zero-width space (​)

Line breaks [p.95] are also white space characters. Note that although   and   are defined in [ISO10646] [p.353] to unambiguously separate lines and paragraphs, respectively, these do not constitute line breaks in HTML, nor does this specification include them in the more general category of white space characters.

This specification does not indicate the behavior, rendering or otherwise, of space characters other than those explicitly identified here as white space characters. For this reason, authors should use appropriate elements and styles to achieve visual formatting effects that involve white space, rather than space characters.

For all HTML elements except PRE, sequences of white space separate "words" (we use the term "word" here to mean "sequences of non-white space characters"). When formatting text, user agents should identify these words and lay them out according to the conventions of the particular written language (script) and target medium.

This layout may involve putting space between words (called *inter-word space*), but conventions for inter-word space vary from script to script. For example, in Latin scripts, inter-word space is typically rendered as an ASCII space (), while in Thai it is a zero-width word separator (​). In Japanese and Chinese, inter-word space is not typically rendered at all.

Note that a sequence of white spaces between words in the source document may result in an entirely different rendered inter-word spacing (except in the case of the PRE element). In particular, user agents should collapse input white space sequences when producing output inter-word space. This can and should be done even in the absence of language information (from the lang attribute, the HTTP "Content-Language" header field (see [RFC2616] [p.354] , section 14.12), user agent settings, etc.).

The PRE element is used for preformatted text [p.97] , where white space is significant.

In order to avoid problems with SGML line break rules [p.335] and inconsistencies among extant implementations, authors should not rely on user agents to render white space immediately after a start tag or immediately before an end tag. Thus, authors, and in particular authoring tools, should write:

```
<P>We offer free <A>technical support</A> for subscribers.</P>
```

and not:

```
<P>We offer free<A> technical support </A>for subscribers.</P>
```

9.2 Structured text

9.2.1 Phrase elements: EM, STRONG, DFN, CODE, SAMP, KBD, VAR, CITE, ABBR, and ACRONYM

```
<!ENTITY % phrase "EM | STRONG | DFN | CODE |
SAMP | KBD | VAR | CITE | ABBR | ACRONYM" >
<!ELEMENT (%fontstyle;|%phrase;) - - (%inline;)* >
<!ATTLIST (%fontstyle;|%phrase;)
%attrs;
-- %coreattrs, %il18n, %events --
>
```

Start tag: required, End tag: required

Attributes defined elsewhere

- `id`, `class` (document-wide identifiers [p.71])
- `lang` (language information [p.79]), `dir` (text direction [p.82])
- `title` (element title [p.63])
- `style` (inline style information [p.186])
- `onclick`, `ondblclick`, `onmousedown`, `onmouseover`, `onmousemove`, `onmouseout`, `onkeypress`, `onkeydown`, `onkeyup` (intrinsic events [p.254])

Phrase elements add structural information to text fragments. The usual meanings of phrase elements are following:

EM: Indicates emphasis.

STRONG: Indicates stronger emphasis.

CITE: Contains a citation or a reference to other sources.

DFN: Indicates that this is the defining instance of the enclosed term.

CODE: Designates a fragment of computer code.

SAMP: Designates sample output from programs, scripts, etc.

KBD: Indicates text to be entered by the user.

VAR: Indicates an instance of a variable or program argument.

ABBR: Indicates an abbreviated form (e.g., WWW, HTTP, URI, Mass., etc.).

ACRONYM: Indicates an acronym (e.g., WAC, radar, etc.).

`EM` and `STRONG` are used to indicate emphasis. The other phrase elements have particular significance in technical documents. These examples illustrate some of the phrase elements:

```
As <CITE>Harry S. Truman</CITE> said,
<Q lang="en-us">The buck stops here.</Q>
More information can be found in <CITE>[ISO-0000]</CITE>.
Please refer to the following reference number in future
correspondence: <STRONG>I-234-55</STRONG>
```

The presentation of phrase elements depends on the user agent. Generally, visual user agents present `EM` text in italics and `STRONG` text in bold font. Speech synthesizer user agents may change the synthesis parameters, such as volume, pitch and rate accordingly.

The `ABBR` and `ACRONYM` elements allow authors to clearly indicate occurrences of abbreviations and acronyms. Western languages make extensive use of acronyms such as "GmbH", "NATO", and "F.B.I.", as well as abbreviations like "M.", "Inc.", "et al.", "etc.". Both Chinese and Japanese use analogous abbreviation mechanisms, wherein a long name is referred to subsequently with a subset of the Han characters from the original occurrence. Marking up these constructs provides useful information to user agents and tools such as spell checkers, speech synthesizers, translation systems and search-engine indexers.

The content of the `ABBR` and `ACRONYM` elements specifies the abbreviated expression itself, as it would normally appear in running text. The title attribute of these elements may be used to provide the full or expanded form of the expression.

Here are some sample uses of `ABBR`:

```
<P>
<ABBR title="World Wide Web">WWW</ABBR>
<ABBR lang="fr"
  title="Sociétéacute;téacute; Nationale des Chemins de Fer">
  SNCF
</ABBR>
<ABBR lang="es" title="Do&ntilde;a">Do&ntilde;a</ABBR>
<ABBR title="Abbreviation">abbr.</ABBR>
```

Note that abbreviations and acronyms often have idiosyncratic pronunciations. For example, while "IRS" and "BBC" are typically pronounced letter by letter, "NATO" and "UNESCO" are pronounced phonetically. Still other abbreviated forms (e.g., "URI" and "SQL") are spelled out by some people and pronounced as words by other people. When necessary, authors should use style sheets to specify the pronunciation of an abbreviated form.

9.2.2 Quotations: The `BLOCKQUOTE` and `Q` elements

```
<!ELEMENT BLOCKQUOTE - - (%block;|SCRIPT)+ -- long quotation -->
<!ATTLIST BLOCKQUOTE
  %attrs;
  cite %URI; #IMPLIED -- URI for source document or msg --
  >
<!ELEMENT Q - - (%inline;)* -- short inline quotation -->
<!ATTLIST Q
  %attrs;
  cite %URI; #IMPLIED -- URI for source document or msg --
  >
```

Start tag: required, End tag: required

Authors specify an object's implementation and the location of the data to be rendered via the `OBJECT` element. To specify run-time values, however, authors use the `PARAM` element, which is discussed in the section on object initialization. [p. 167]

The `OBJECT` element may also appear in the content of the `HEAD` element. Since user agents generally do not render elements in the `HEAD`, authors should ensure that any `OBJECT` elements in the `HEAD` do not specify content that may be rendered. Please consult the section on sharing frame data [p.208] for an example of including the `OBJECT` element in the `HEAD` element.

Please consult the section on form controls [p.220] for information about `OBJECT` elements in forms.

This document does not specify the behavior of `OBJECT` elements that use both the `classid` attribute to identify an implementation and the `data` attribute to specify data for that implementation. In order to ensure portability, authors should use the `PARAM` element to tell implementations where to retrieve additional data.

13.3.1 Rules for rendering objects

A user agent must interpret an `OBJECT` element according to the following precedence rules:

1. The user agent must first try to render the object. It should not render the element's contents, but it must examine them in case the element contains any direct children that are `PARAM` elements (see object initialization [p.167]) or `MAP` elements (see client-side image maps [p.174]).
2. If the user agent is not able to render the object for whatever reason (configured not to, lack of resources, wrong architecture, etc.), it must try to render its contents.

Authors should not include content in `OBJECT` elements that appear in the `HEAD` element.

In the following example, we insert an analog clock applet in a document via the `OBJECT` element. The applet, written in the Python language, requires no additional data or run-time values. The `classid` attribute specifies the location of the applet:

```
<P><OBJECT classid="http://www.miamachina.it/analogclock.py">
</OBJECT>
```

Note that the clock will be rendered as soon as the user agent interprets this `OBJECT` declaration. It is possible to delay rendering of an object by first declaring the object (described below).

Authors should complete this declaration by including alternate text as the contents of `OBJECT` in case the user agent cannot render the clock.

```
<P><OBJECT classid="http://www.miamachina.it/analogclock.py">
An animated clock.
</OBJECT>
```

One significant consequence of the `OBJECT` element's design is that it offers a mechanism for specifying alternate object renderings; each embedded `OBJECT` declaration may specify alternate content types. If a user agent cannot render the outermost `OBJECT`, it tries to render the contents, which may be another `OBJECT` element, etc.

In the following example, we embed several `OBJECT` declarations to illustrate how alternate renderings work. A user agent will attempt to render the first `OBJECT` element it can, in the following order: (1) an Earth applet written in the Python language, (2) an MPEG animation of the Earth, (3) a GIF image of the Earth, (4) alternate text.

```
<P>
<!-- First, try the Python applet -->
<OBJECT title="The Earth as seen from space"
classid="http://www.observer.mars/TheEarth.py">
<!-- Else, try the MPEG video -->
<OBJECT data="TheEarth.mpeg" type="application/mpeg">
<!-- Else, try the GIF image -->
<OBJECT data="TheEarth.gif" type="image/gif">
<!-- Else render the text -->
The <STRONG>Earth</STRONG> as seen from space.
</OBJECT>
</OBJECT>
</OBJECT>
```

The outermost declaration specifies an applet that requires no data or initial values. The second declaration specifies an MPEG animation and, since it does not define the location of an implementation to handle MPEG, relies on the user agent to handle the animation. We also set the `type` attribute so that a user agent that knows it cannot render MPEG will not bother to retrieve "TheEarth.mpeg" from the network. The third declaration specifies the location of a GIF file and furnishes alternate text in case all other mechanisms fail.

Inline vs. external data. Data to be rendered may be supplied in two ways: *inline* and *from an external resource*. While the former method will generally lead to faster rendering, it is not convenient when rendering large quantities of data.

Here's an example that illustrates how inline data may be fed to an `OBJECT`:

```
<P>
<OBJECT id="clock1"
classid="clsid:663C8FEF-1EF9-11CF-A3DB-080036F12502"
data="data:application/x-oleobject;base64, ...base64 data...">
A clock.
</OBJECT>
```

Please consult the section on the visual presentation of objects, images, and applets [p. 179] for information about object size, alignment, and borders.

13.3.2 Object initialization: the PARAM element

```
<!ELEMENT PARAM - O EMPTY
<!ATTLIST PARAM
  id ID #IMPLIED -- document-wide unique id --
  name CDATA #REQUIRED -- property name --
  value CDATA #IMPLIED -- property value --
  valueType (DATA|REF|OBJECT) DATA -- How to interpret value --
  type %ContentType; #IMPLIED -- content type for value
  when valueType=ref --
  >
```

Start tag: required, End tag: forbidden

Attribute definitions

`name = cdata` [p.50]

This attribute defines the name of a run-time parameter, assumed to be known by the inserted object. Whether the property name is case-sensitive depends on the specific object implementation.

`value = cdata` [p.50]

This attribute specifies the value of a run-time parameter specified by the `name`. Property values have no meaning to HTML; their meaning is determined by the object in question.

`valueType = data|ref|object [CI]` [p.49]

This attribute specifies the type of the `value` attribute. Possible values:

- `data`: This is default value for the attribute. It means that the value specified by `value` will be evaluated and passed to the object's implementation as a string.
- `ref`: The value specified by `value` is a URI that designates a resource where run-time values are stored. This allows support tools to identify URIs given as parameters. The URI must be passed to the object as **is**, i.e., unresolved.
- `object`: The value specified by `value` is an identifier that refers to an OBJECT declaration in the same document. The identifier must be the value of the `id` attribute set for the declared OBJECT element.

`type = content-type [p.53] [CI]` [p.49]

This attribute specifies the content type of the resource designated by the `value` attribute **only** in the case where `valueType` is set to "ref". This attribute thus specifies for the user agent, the type of values that will be found at the URI designated by `value`.

Attributes defined elsewhere

- `id` (document-wide identifiers [p.71])

PARAM elements specify a set of values that may be required by an object at run-time. Any number of PARAM elements may appear in the content of an OBJECT or APPLET element, in any order, but must be placed at the start of the content of the enclosing OBJECT or APPLET element.

The syntax of names and values is assumed to be understood by the object's implementation. This document does not specify how user agents should retrieve name/value pairs nor how they should interpret parameter names that appear twice.

We return to the clock example to illustrate the use of PARAM: suppose that the applet is able to handle two run-time parameters that define its initial height and width. We can set the initial dimensions to 40x40 pixels with two PARAM elements.

```
<P><OBJECT classid="http://www.miamachina.it/analogslook.py">
  <PARAM name="height" value="40" valueType="data">
  <PARAM name="width" value="40" valueType="data">
  This user agent cannot render Python applications.
</OBJECT>
```

In the following example, run-time data for the object's "init_values" parameter is specified as an external resource (a GIF file). The value of the valueType attribute is thus set to "ref" and the value is a URI designating the resource.

```
<P><OBJECT classid="http://www.gifstuff.com/gifapplet"
  standby="Loading Elvis...">
  <PARAM name="Init_values"
    values="/images/elvis.gif"
    valueType="ref">
</OBJECT>
```

Note that we have also set the `standby` attribute so that the user agent may display a message while the rendering mechanism loads.

When an OBJECT element is rendered, user agents must search the content for only those PARAM elements that are direct children and "feed" them to the OBJECT.

Thus, in the following example, if "obj1" is rendered, "param1" applies to "obj1" (and not "obj2"). If "obj1" is not rendered and "obj2" is, "param1" is ignored, and "param2" applies to "obj2". If neither OBJECT is rendered, neither PARAM applies.

```
<P>
  <OBJECT id="obj1">
    <PARAM name="param1">
  </OBJECT id="obj2">
    <PARAM name="param2">
  </OBJECT>
</OBJECT>
```

13.3.3 Global naming schemes for objects

The location of an object's implementation is given by a URI. As we discussed in the introduction to URIs [p. 19], the first segment of an absolute URI specifies the naming scheme used to transfer the data designated by the URI. For HTML documents, this scheme is frequently "http". Some applets might employ other naming schemes. For instance, when specifying a Java applet, authors may use URIs that begin with "java" and for ActiveX applets, authors may use "clsid".

In the following example, we insert a Java applet into an HTML document.

```
<P><OBJECT classid="java:program.start" >
</OBJECT>
```

By setting the `codetype` attribute, a user agent can decide whether to retrieve the Java application based on its ability to do so.

```
<OBJECT codetypes="application/java-archive"
classid="java:program.start" >
</OBJECT>
```

Some rendering schemes require additional information to identify their implementation and must be told where to find that information. Authors may give path information to the object's implementation via the `codebase` attribute.

```
<OBJECT codetypes="application/java-archive"
classid="java:program.start" >
codebase="http://foooo.bar.com/java/myimplementation/"
</OBJECT>
```

The following example specifies (with the `classid` attribute) an ActiveX object via a URI that begins with the naming scheme "clsid". The `data` attribute locates the data to render (another clock).

```
<P><OBJECT classid="clsid:663C8F8F-1EF9-11CF-A3DB-080036F12502"
data="http://www.acme.com/ole/clock.stm" >
This application is not supported.
</OBJECT>
```

13.3.4 Object declarations and instantiations

The preceding examples have only illustrated isolated object definitions. When a document is to contain more than one instance of the same object, it is possible to separate the declaration of the object from its instantiations. Doing so has several advantages:

- Data may be retrieved from the network by the user agent *one time* (during the declaration) and reused for each instantiation.
- It is possible to instantiate an object from a location other than the object's declaration, for example, from a link.
- It is possible to specify objects as run-time data for other objects.

To declare an object so that it is not executed when read by the user agent, set the boolean `declare` attribute in the `OBJECT` element. At the same time, authors must identify the declaration by setting the `id` attribute in the `OBJECT` element to a unique value. Later instantiations of the object will refer to this identifier.

A declared `OBJECT` must appear in a document before the first instance of that `OBJECT`.

An object defined with the `declare` attribute is instantiated every time an element that refers to that object requires it to be rendered (e.g., a link that refers to it is activated, an object that refers to it is activated, etc.).

In the following example, we declare an `OBJECT` and cause it to be instantiated by referring to it from a link. Thus, the object can be activated by clicking on some highlighted text, for example.

```
<P><OBJECT declare
id="earth.declaration"
data="TheEarth.mpeg"
type="application/mpeg" >
The <STRONG>Earth</STRONG> as seen from space.
</OBJECT>
...later in the document...
<P>A neat <A href="#earth.declaration"> animation of The Earth!</A>
```

The following example illustrates how to specify run-time values that are other objects. In this example, we send text (a poem, in fact) to a hypothetical mechanism for viewing poems. The object recognizes a run-time parameter named "font" (say, for rendering the poem text in a certain font). The value for this parameter is itself an object that inserts (but does not render) the font object. The relationship between the font object and the poem viewer object is achieved by (1) assigning the `id` "tribune" to the font object declaration and (2) referring to it from the `PARAM` element of the poem viewer object (with `valueType` and `value`).

```
<P><OBJECT declare
id="tribune"
type="application/x-webfont"
data="tribune.gif" >
</OBJECT>
...view the poem in Kublakhan.txt here...
<P><OBJECT classid="http://foo.bar.com/poem_viewer"
data="Kublakhan.txt" >
<PARAM name="font" valueType="object" value="#tribune" >
<P>You're missing a really cool poem viewer ...
</OBJECT>
```

User agents that don't support the `declare` attribute must render the contents of the `OBJECT` declaration.

13.4 Including an applet: the `APPLET` element

APPLET is deprecated (with all its attributes) [p.38] in favor of `OBJECT`.

See the Transitional DTD [p.286] for the formal definition.

Attribute definitions

`codebase = uri` [p.51] [CT] [p.49]

This attribute specifies the base URI for the applet. If this attribute is not specified, then it defaults the same base URI as for the current document. Values for this attribute may only refer to subdirectories of the directory containing the current document. **Note.** While the restriction on subdirectories is a departure from common practice and the HTML 3.2 specification, the HTML Working Group has chosen to leave the restriction in this version of the specification for security reasons.

`code = cdata` [p.50] [CS] [p.49]

This attribute specifies either the name of the class file that contains the applet's compiled applet subclass or the path to get the class, including the class file itself. It is interpreted with respect to the applet's codebase. One of `code` or `object` must be present.

`name = cdata` [p.50] [CS] [p.49]

This attribute specifies a name for the applet instance, which makes it possible for applets on the same page to find (and communicate with) each other.

`archive = uri-list` [p.51] [CT] [p.49]

This attribute specifies a comma-separated list of URIs for archives containing classes and other resources that will be "preloaded". The classes are loaded using an instance of an `AppletClassLoader` with the given codebase. Relative URIs for archives are interpreted with respect to the applet's codebase.

Preloading resources can significantly improve the performance of applets.

`object = cdata` [p.50] [CS] [p.49]

This attribute names a resource containing a serialized representation of an applet's state. It is interpreted relative to the applet's codebase. The serialized data contains the applet's class name but not the implementation. The class name is used to retrieve the implementation from a class file or archive.

When the applet is "deserialized" the `start()` method is invoked but not the `init()` method. Attributes valid when the original object was serialized are **not** restored. Any attributes passed to this `APPLET` instance will be available to the applet. Authors should use this feature with extreme caution. An applet should be stopped before it is serialized.

Either `code` or `object` must be present. If both `code` and `object` are given, it is an error if they provide different class names.

`width = length` [p.52] [CI] [p.49]

This attribute specifies the initial width of the applet's display area (excluding any windows or dialogs that the applet creates).

`height = length` [p.52] [CI] [p.49]

This attribute specifies the initial height of the applet's display area (excluding any windows or dialogs that the applet creates).

Attributes defined elsewhere

- `id`, `class` (document-wide identifiers [p.71])
- `title` (element title [p.63])
- `style` (inline style information [p.186])
- `alt` (alternate text [p.181])
- `align`, `hspace`, `vspace` (visual presentation of objects, images, and applets [p.179])

This element, supported by all Java-enabled browsers, allows designers to embed a Java applet in an HTML document. It has been deprecated [p.38] in favor of the `OBJECT` element.

The content of the `APPLET` acts as alternate information for user agents that don't support this element or are currently configured not to support applets. User agents must ignore the content otherwise.

DEPRECATED EXAMPLE:

In the following example, the `APPLET` element includes a Java applet in the document. Since no codebase is supplied, the applet is assumed to be in the same directory as the current document.

```
<APPLET code="Bubbles.class" width="500" height="500">
Java applet that draws animated bubbles.
</APPLET>
```

This example may be rewritten with `OBJECT` as follows:

```
<P><OBJECT codetype="application/java"
classid="Java:Bubbles.class"
width="500" height="500">
Java applet that draws animated bubbles.
</OBJECT>
```

Initial values may be supplied to the applet via the `PARAM` element.

DEPRECATED EXAMPLE:

The following sample Java applet:

```
<APPLET code="AudioItem" width="15" height="15">
<PARAM name="snd" value="Hello.au|Welcome.au">
Java applet that plays a welcoming sound.
</APPLET>
```

may be rewritten as follows with `OBJECT`:

```
<OBJECT codetype="application/java"
  classid="AudioItem"
  width="15" height="15">
<PARAM name="snd" value="Hello.au|welcome.au">
Java applet that plays a welcoming sound.
</OBJECT>
```

13.5 Notes on embedded documents

Sometimes, rather than linking [p.145] to a document, an author may want to embed it directly into a primary HTML document. Authors may use either the `IFRAME` element or the `OBJECT` element for this purpose, but the elements differ in some ways. Not only do the two elements have different content models, the `IFRAME` element may be a target frame (see the section on specifying target frame information [p.212] for details) and may be "selected" by a user agent as the focus for printing, viewing HTML source, etc. User agents may render selected frames elements in ways that distinguish them from unselected frames (e.g., by drawing a border around the selected frame).

An embedded document is entirely independent of the document in which it is embedded. For instance, relative URIs within the embedded document resolve [p.158] according to the base URI of the embedded document, not that of the main document. An embedded document is only rendered within another document (e.g., in a subwindow); it remains otherwise independent.

For instance, the following line embeds the contents of `embed_me.html` at the location where the `OBJECT` definition occurs.

```
...text before...
<OBJECT data="embed_me.html">
Warning: embed_me.html could not be embedded.
</OBJECT>
...text after...
```

Recall that the contents of `OBJECT` must only be rendered if the file specified by the `data` attribute cannot be loaded.

The behavior of a user agent in cases where a file includes itself is not defined.

13.6 Image maps

Image maps allow authors to specify regions of an image or object and assign a specific action to each region (e.g., retrieve a document, run a program, etc.) When the region is activated by the user, the action is executed.

An image map is created by associating an object with a specification of sensitive geometric areas on the object.

There are two types of image maps:

- **Client-side.** When a user activates a region of a client-side image map with a mouse, the pixel coordinates are interpreted by the user agent. The user agent selects a link that was specified for the activated region and follows it.
- **Server-side.** When a user activates a region of a server-side image map with a mouse, the pixel coordinates of the click are sent to the server-side agent specified by the `href` attribute of the `A` element. The server-side agent interprets the coordinates and performs some action.

Client-side image maps are preferred over server-side image maps for at least two reasons: they are accessible to people browsing with non-graphical user agents and they offer immediate feedback as to whether or not the pointer is over an active region.

13.6.1 Client-side image maps: the `MAP` and `AREA` elements

```
<!ELEMENT MAP - - ((%block;) | AREA)+ -- client-side image map -->
<!ATTLIST MAP
  %attrs;
  name          CDATA          #REQUIRED -- for reference by usemap --
  >
```

Start tag: required, End tag: required

```
<!ELEMENT AREA - O EMPTY
<!ATTLIST AREA
  %attrs;
  shape          %Shape;
  coords         %Coords;
  href           %URI;
  nohref         (nohref)
  alt            %Text;
  tabindex       NUMBER
  accesskey     %Character;
  onfocus       %Script;
  onblur        %Script;
  >
  rect          -- controls interpretation of coords --
  HIMPLIED    -- comma-separated list of lengths --
  HIMPLIED    -- URI for linked resource --
  HIMPLIED    -- this region has no action --
  HREQUIRED  -- short description --
  HIMPLIED    -- position in tabbing order --
  HIMPLIED    -- accessibility key character --
  HIMPLIED    -- the element got the focus --
  HIMPLIED    -- the element lost the focus --
  >
```

Start tag: required, End tag: forbidden

MAP attribute definitions

`name = cdata [p.50] [C] [p.49]`

This attribute assigns a name to the image map defined by a `MAP` element.

AREA attribute definitions

`shape = default | rect | circle | poly [C] [p.49]`

This attribute specifies the shape of a region. Possible values:

- `default`: Specifies the entire region.
- `rect`: Define a rectangular region.
- `circle`: Define a circular region.
- `poly`: Define a polygonal region.

`coords = coordinates [CN] [p.49]`

This attribute specifies the position and shape on the screen. The number and order of values depends on the shape being defined. Possible combinations:

- `rect`: left-x, top-y, right-x, bottom-y.
- `circle`: center-x, center-y, radius. **Note.** When the radius value is a percentage value, user agents should calculate the final radius value based on the associated object's width and height. The radius should be the smaller value of the two.
- `poly`: x1, y1, x2, y2, ..., xN, yN. The first x and y coordinate pair and the last should be the same to close the polygon. When these coordinate values are not the same, user agents should infer an additional coordinate pair to close the polygon.

Coordinates are relative to the top, left corner of the object. All values are lengths [p.52]. All values are separated by commas.

`nohref [C] [p.49]`

When set, this boolean attribute specifies that a region has no associated link.

Attribute to associate an image map with an element

`usemap = uri [p.51] [CT] [p.49]`

This attribute associates an image map with an element. The image map is defined by a `MAP` element. The value of `usemap` must match the value of the name attribute of the associated `MAP` element.

Attributes defined elsewhere

- `id, class` (document-wide identifiers [p.71])
- `lang` (language information [p.79]), `dir` (text direction [p.82])
- `title` (element title [p.63])
- `style` (inline style information [p.186])
- `name` (submitting objects with forms [p.245])
- `alt` (alternate text [p.181])
- `href` (anchor reference [p.149]) `target` (frame target information [p.212])
- `tabindex` (tabbing navigation [p.241])
- `accesskey` (access keys [p.242])
- `shape` (image maps [p.173])
- `onclick, ondblclick, onmousedown, onmouseup, onmouseover, onmousemove, onmouseout, onkeypress, onkeydown, onkeyup, onfocus, onblur` (intrinsic events [p.254])

The `MAP` element specifies a client-side image map (or other navigation mechanism) that may be associated with another elements (`IMG`, `OBJECT`, or `INPUT`). An image map is associated with an element via the element's `usemap` attribute. The `MAP` element may be used without an associated image for general navigation mechanisms.

The presence of the `usemap` attribute for an `OBJECT` implies that the object being included is an image. Furthermore, when the `OBJECT` element has an associated client-side image map, user agents may implement user interaction with the `OBJECT` solely in terms of the client-side image map. This allows user agents (such as an audio browser or robot) to interact with the `OBJECT` without having to process it; the user agent may even elect not to retrieve (or process) the object. When an `OBJECT` has an associated image map, authors should not expect that the object will be retrieved or processed by every user agent.

The `MAP` element content model allows authors to combine the following:

1. One or more `AREA` elements. These elements have no content but specify the geometric regions of the image map and the link associated with each region. Note that user agents do not generally render `AREA` elements. Therefore, authors must provide alternate text for each `AREA` with the `alt` attribute (see below for information on how to specify alternate text [p.181]).
2. Block-level content. This content should include `A` elements that specify the geometric regions of the image map and the link associated with each region. Note that the user agent should render block-level content of a `MAP` element. Authors should use this method to create more accessible documents.

When a `MAP` element contains mixed content (both `AREA` elements and block-level content), user agents must ignore the `AREA` elements.

Authors should specify an image map's geometry completely with `AREA` elements, or completely with `A` elements, or completely with both if content is mixed. Authors may wish to mix content so that older user agents will handle map geometries specified by `AREA` elements and new user agents will take advantage of richer block content.

If two or more defined regions overlap, the region-defining element that appears earliest in the document takes precedence (i.e., responds to user input).

User agents and authors should offer textual alternates to graphical image maps for cases when graphics are not available or the user cannot access them. For example, user agents may use `alt` text to create textual links in place of a graphical image map. Such links may be activated in a variety of ways (keyboard, voice activation, etc.).

Note. *MAP is not backwards compatible with HTML 2.0 user agents.*

Client-side image map examples

In the following example, we create a client-side image map for the `OBJECT` element. We do not want to render the image map's contents when the `OBJECT` is rendered, so we "hide" the `MAP` element within the `OBJECT` element's content. Consequently, the `MAP` element's contents will only be rendered if the `OBJECT` cannot be rendered.


```

<HTML>
<HEAD>
<TITLE>The cool site!</TITLE>
</HEAD>
<BODY>
<P><OBJECT data="navbar1.gif" type="image/gif" usemap="#map1">
<MAP name="map1">
<P>Navigate the site:
<A href="guide.html" shape="rect" coords="0,0,118,28">Access Guide</a> |
<A href="shortcut.html" shape="rect" coords="118,0,184,28">Go</a> |
<A href="search.html" shape="circle" coords="184,200,60">Search</a> |
<A href="top10.html" shape="poly" coords="276,0,276,28,100,200,50,276,0">Top Ten</a>
</MAP>
</OBJECT>
</BODY>
</HTML>

```

We may want to render the image map's contents even when a user agent can render the OBJECT. For instance, we may want to associate an image map with an OBJECT element *and* include a text navigation bar at the bottom of the page. To do so, we define the MAP element outside the OBJECT:

```

<HTML>
<HEAD>
<TITLE>The cool site!</TITLE>
</HEAD>
<BODY>
<P><OBJECT data="navbar1.gif" type="image/gif" usemap="#map1">
</OBJECT>
...the rest of the page here...
<MAP name="map1">
<P>Navigate the site:
<A href="guide.html" shape="rect" coords="0,0,118,28">Access Guide</a> |
<A href="shortcut.html" shape="rect" coords="118,0,184,28">Go</a> |
<A href="search.html" shape="circle" coords="184,200,60">Search</a> |
<A href="top10.html" shape="poly" coords="276,0,276,28,100,200,50,276,0">Top Ten</a>
</MAP>
</BODY>
</HTML>

```

In the following example, we create a similar image map, this time using the AREA element. Note the use of alt text:

```

<P><OBJECT data="navbar1.gif" type="image/gif" usemap="#map1">
<P>This is a navigation bar.
</OBJECT>

<MAP name="map1">
<AREA href="guide.html"
alt="Access Guide"
shape="rect"
coords="0,0,118,28">
<AREA href="search.html"
alt="Search"
shape="rect"
coords="184,0,276,28">
alt="Go"
<AREA href="shortcut.html"
shape="circle"
coords="184,200,60">
<AREA href="top10.html"

```

```

alt="Top Ten"
shape="poly"
coords="276,0,276,28,100,200,50,276,0">
</MAP>

```

Here is a similar version using the IMG element instead of OBJECT (with the same MAP declaration):

```

<P><IMG src="navbar1.gif" usemap="#map1" alt="navigation bar">

```

The following example illustrates how image maps may be shared.

Nested OBJECT elements are useful for providing fallbacks in case a user agent doesn't support certain formats. For example:

```

<P>
<OBJECT data="navbar.png" type="image/png">
<OBJECT data="navbar.gif" type="image/gif">
text describing the image...
</OBJECT>
</OBJECT>

```

If the user agent doesn't support the PNG format, it tries to render the GIF image. If it doesn't support GIF (e.g., it's a speech-based user agent), it defaults to the text description provided as the content of the inner OBJECT element. When OBJECT elements are nested this way, authors may share image maps among them:

```

<P>
<OBJECT data="navbar.png" type="image/png" usemap="#map1">
<OBJECT data="navbar.gif" type="image/gif" usemap="#map1">
<MAP name="map1">
<P>Navigate the site:
<A href="guide.html" shape="rect" coords="0,0,118,28">Access Guide</a> |
<A href="shortcut.html" shape="rect" coords="118,0,184,28">Go</a> |
<A href="search.html" shape="circle" coords="184,200,60">Search</a> |
<A href="top10.html" shape="poly" coords="276,0,276,28,100,200,50,276,0">Top Ten</a>
</MAP>
</OBJECT>
</OBJECT>

```

The following example illustrates how anchors may be specified to create inactive zones within an image map. The first anchor specifies a small circular region with no associated link. The second anchor specifies a larger circular region with the same center coordinates. Combined, the two form a ring whose center is inactive and whose rim is active. The order of the anchor definitions is important, since the smaller circle must override the larger circle.

```

<MAP name="map1">
<P>
<A shape="circle" coords="100,200,50">I'm inactive.</A>
<A href="outer-ring-link.html" shape="circle" coords="100,200,250">I'm active.</A>
</MAP>

```

Similarly, the nohref attribute for the AREA element declares that geometric region has no associated link.

13.6.2 Server-side image maps

Server-side image maps may be interesting in cases where the image map is too complicated for a client-side image map.

It is only possible to define a server-side image map for the `IMG` and `INPUT` elements. In the case of `IMG`, the `IMG` must be inside an `A` element and the boolean attribute `ismap` (`[C]` [p.49]) must be set. In the case of `INPUT`, the `INPUT` must be of type "image".

When the user activates the link by clicking on the image, the screen coordinates are sent directly to the server where the document resides. Screen coordinates are expressed as screen pixel values relative to the image. For normative information about the definition of a pixel and how to scale it, please consult [CSS'1] [p.353].

In the following example, the active region defines a server-side link. Thus, a click anywhere on the image will cause the click's coordinates to be sent to the server.

```
<P><A href="http://www.acme.com/cgi-bin/competition" >
  <IMG src="game.gif" ismap alt="target"></A>
```

The location clicked is passed to the server as follows. The user agent derives a new URI from the URI specified by the `href` attribute of the `A` element, by appending "?" followed by the x and y coordinates, separated by a comma. The link is then followed using the new URI. For instance, in the given example, if the user clicks at the location x=10, y=27 then the derived URI is "http://www.acme.com/cgi-bin/competition?10,27".

User agents that do not offer the user a means to select specific coordinates (e.g., non-graphical user agents that rely on keyboard input, speech-based user agents, etc.) should send the coordinates "0,0" to the server when the link is activated.

13.7 Visual presentation of images, objects, and applets

All `IMG` and `OBJECT` attributes that concern visual alignment and presentation have been deprecated [p.38] in favor of style sheets.

13.7.1 Width and height

Attribute definitions

```
width = length [p.52] [CN] [p.49]
  Image and object width override.
height = length [p.52] [CN] [p.49]
  Image and object height override.
```

When specified, the `width` and `height` attributes tell user agents to override the natural image or object size in favor of these values.

When the object is an image, it is scaled. User agents should do their best to scale an object or image to match the width and height specified by the author. Note that lengths expressed as percentages are based on the horizontal or vertical space currently available, not on the natural size of the image, object, or applet.

The `height` and `width` attributes give user agents an idea of the size of an image or object so that they may reserve space for it and continue rendering the document while waiting for the image data.

13.7.2 White space around images and objects

Attribute definitions

```
hspace = pixels [p.52] [CN] [p.49]
```

Deprecated. [p.38] This attribute specifies the amount of white space to be inserted to the left and right of an `IMG`, `APPLET`, or `OBJECT`. The default value is not specified, but is generally a small, non-zero length.

```
vspace = pixels [p.52] [CN] [p.49]
```

Deprecated. [p.38] This attribute specifies the amount of white space to be inserted above and below an `IMG`, `APPLET`, or `OBJECT`. The default value is not specified, but is generally a small, non-zero length.

13.7.3 Borders

An image or object may be surrounded by a border (e.g., when a border is specified by the user or when the image is the content of an `A` element).

Attribute definitions

```
border = pixels [p.52] [CN] [p.49]
```

Deprecated. [p.38] This attribute specifies the width of an `IMG` or `OBJECT` border, in pixels. The default value for this attribute depends on the user agent.

13.7.4 Alignment

Attribute definitions

```
align = bottom|middle|top|left|right
```

Deprecated. [p.38] This attribute specifies the position of an `IMG`, `OBJECT`, or `APPLET` with respect to its context.

The following values for `align` concern the object's position with respect to surrounding text:

- `bottom`: means that the bottom of the object should be vertically aligned with the current baseline. This is the default value.

- `middle`: means that the center of the object should be vertically aligned with the current baseline.
- `top`: means that the top of the object should be vertically aligned with the top of the current text line.

Two other values, `left` and `right`, cause the image to float to the current left or right margin. They are discussed in the section on floating objects [p.197] .

Differing interpretations of align. *User agents vary in their interpretation of the align attribute. Some only take into account what has occurred on the text line prior to the element, some take into account the text on both sides of the element.*

13.8 How to specify alternate text

Attribute definitions

`alt = text` [p.50] [CS] [p.49]

For user agents that cannot display images, forms, or applets, this attribute specifies alternate text. The language of the alternate text is specified by the `lang` attribute.

Several non-textual elements (`IMG`, `AREA`, `APPLET`, and `INPUT`) let authors specify alternate text to serve as content when the element cannot be rendered normally. Specifying alternate text assists users without graphic display terminals, users whose browsers don't support forms, visually impaired users, those who use speech synthesizers, those who have configured their graphical user agents not to display images, etc.

The `alt` attribute must be specified for the `IMG` and `AREA` elements. It is optional for the `INPUT` and `APPLET` elements.

While alternate text may be very helpful, it must be handled with care. Authors should observe the following guidelines:

- Do not specify irrelevant alternate text when including images intended to *format* a page, for instance, `alt="red ball"` would be inappropriate for an image that adds a red ball for decorating a heading or paragraph. In such cases, the alternate text should be the empty string (""). Authors are in any case advised to avoid using images to format pages; style sheets should be used instead.
- Do not specify meaningless alternate text (e.g., "dummy text"). Not only will this frustrate users, it will slow down user agents that must convert text to speech or braille output.

Implementors should consult the section on accessibility [p.350] for information about how to handle cases of omitted alternate text.

14 Style Sheets

Contents

1. Introduction to style sheets	183
2. Adding style to HTML	185
1. Setting the default style sheet language	186
2. Inline style information	186
3. Header style information: the <code>STYLE</code> element	187
4. Media types	189
3. External style sheets	190
1. Preferred and alternate style sheets	190
2. Specifying external style sheets	191
4. Cascading style sheets	192
1. Media-dependent cascades	192
2. Inheritance and cascading	193
5. Hiding style data from user agents	193
6. Linking to style sheets with HTTP headers	194

14.1 Introduction to style sheets

Style sheets represent a major breakthrough for Web page designers, expanding their ability to improve the appearance of their pages. In the scientific environments in which the Web was conceived, people are more concerned with the content of their documents than the presentation. As people from wider walks of life discovered the Web, the limitations of HTML became a source of continuing frustration and authors were forced to sidestep HTML's stylistic limitations. While the intentions have been good -- to improve the presentation of Web pages -- the techniques for doing so have had unfortunate side effects. These techniques work for some of the people, some of the time, but not for all of the people, all of the time. They include:

- Using proprietary HTML extensions
- Converting text into images
- Using images for white space control
- Use of tables for page layout
- Writing a program instead of using HTML

These techniques considerably increase the complexity of Web pages, offer limited flexibility, suffer from interoperability problems, and create hardships for people with disabilities.

Style sheets solve these problems at the same time they supersede the limited range of presentation mechanisms in HTML. Style sheets make it easy to specify the amount of white space between text lines, the amount lines are indented, the colors used for the text and the backgrounds, the font size and style, and a host of other details.

For example, the following short CSS style sheet (stored in the file "special.css"), sets the text color of a paragraph to green and surrounds it with a solid red border:

```
P.special {
  color : green;
  border: solid red;
}
```

Authors may link this style sheet to their source HTML document with the `LINK` element:

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01//EN"
"http://www.w3.org/TR/html4/strict.dtd">
<HTML>
<HEAD>
<LINK href="special.css" rel="stylesheet" type="text/css">
</HEAD>
<P class="special">This paragraph should have special green text.
</BODY>
</HTML>
```

HTML 4 provides support for the following style sheet features:

Flexible placement of style information

Placing style sheets in separate files makes them easy to reuse. Sometimes it's useful to include rendering instructions within the document to which they apply, either grouped at the start of the document, or in attributes of the elements throughout the body of the document. To make it easier to manage style on a site basis, this specification describes how to use HTTP headers to set the style sheets to be applied to a document.

Independence from specific style sheet languages

This specification doesn't tie HTML to any particular style sheet language. This allows for a range of such languages to be used, for instance simple ones for the majority of users and much more complex ones for the minority of users with highly specialized needs. The examples included below all use the CSS (Cascading Style Sheets) language [CSS1] [p.353], but other style sheet languages would be possible.

Cascading

This is the capability provided by some style sheet languages such as CSS to allow style information from several sources to be blended together. These could be, for instance, corporate style guidelines, styles common to a group of documents, and styles specific to a single document. By storing these separately, style sheets can be reused, simplifying authoring and making more effective use of network caching. The cascade defines an ordered sequence of style sheets where rules in later sheets have greater precedence than earlier ones. Not all style sheet languages support cascading.

Media dependencies

HTML allows authors to specify documents in a media-independent way. This allows users to access Web pages using a wide variety of devices and media,

e.g., graphical displays for computers running Windows, Macintosh OS, and X11, devices for television sets, specially adapted phones and PDA-based portable devices, speech-based browsers, and braille-based tactile devices.

Style sheets, by contrast, apply to specific media or media groups. A style sheet intended for screen use may be applicable when printing, but is of little use for speech-based browsers. This specification allows you to define the broad categories of media a given style sheet is applicable to. This allows user agents to avoid retrieving inappropriate style sheets. Style sheet languages may include features for describing media dependencies within the same style sheet.

Alternate styles

Authors may wish to offer readers several ways to view a document. For instance, a style sheet for rendering compact documents with small fonts, or one that specifies larger fonts for increased legibility. This specification allows authors to specify a preferred style sheet as well as alternates that target specific users or media. User agents should give users the opportunity to select from among alternate style sheets or to switch off style sheets altogether.

Performance concerns

Some people have voiced concerns over performance issues for style sheets. For instance, retrieving an external style sheet may delay the full presentation for the user. A similar situation arises if the document head includes a lengthy set of style rules.

The current proposal addresses these issues by allowing authors to include rendering instructions within each HTML element. The rendering information is then always available by the time the user agent wants to render each element.

In many cases, authors will take advantage of a common style sheet for a group of documents. In this case, distributing style rules throughout the document will actually lead to worse performance than using a linked style sheet, since for most documents, the style sheet will already be present in the local cache. The public availability of good style sheets will encourage this effect.

14.2 Adding style to HTML

Note. *The sample default style sheet for HTML 4 that is included in [CSS2] [p.355] expresses generally accepted default style information for each element. Authors and implementors alike might find this a useful resource.*

HTML documents may contain style sheet rules directly in them or they may import style sheets.

Any style sheet language may be used with HTML. A simple style sheet language may suffice for the needs of most users, but other languages may be more suited to highly specialized needs. This specification uses the style language "Cascading Style Sheets" ([CSS1] [p.353]), abbreviated CSS, for examples.

The syntax of style data [p.57] depends on the style sheet language.

14.2.1 Setting the default style sheet language

Authors must specify the style sheet language of style information associated with an HTML document.

Authors should use the META element to set the default style sheet language for a document. For example, to set the default to CSS, authors should put the following declaration in the HEAD of their documents:

```
<META http-equiv="Content-Style-Type" content="text/css">
```

The default style sheet language may also be set with HTTP headers. The above META declaration is equivalent to the HTTP header:

```
Content-Style-Type: text/css
```

User agents should determine the default style sheet language for a document according to the following steps (highest to lowest priority):

1. If any META declarations specify the "Content-Style-Type", the last one in the character stream determines the default style sheet language.
2. Otherwise, if any HTTP headers specify the "Content-Style-Type", the last one in the character stream determines the default style sheet language.
3. Otherwise, the default style sheet language is "text/css".

Documents that include elements that set the style attribute but which don't define a default style sheet language are incorrect. Authoring tools should generate default style sheet language information (typically a META declaration) so that user agents do not have to rely on a default of "text/css".

14.2.2 Inline style information

Attribute definitions

`style = style` [p.57] [CN] [p.49]

This attribute specifies style information for the current element.

The syntax [p.57] of the value of the style attribute is determined by the default style sheet language [p.186]. For example, for [[CSS2]] inline style, use the declaration block syntax described in section 4.1.8 (without curly brace delimiters).

This CSS example sets color and font size information for the text in a specific paragraph.

```
<p style="font-size: 12pt; color: fuchsia">Aren't style sheets wonderful?
```

In CSS, property declarations have the form "name : value" and are separated by a semi-colon.

To specify style information for more than one element, authors should use the `STYLE` element. For optimal flexibility, authors should define styles in external style sheets.

14.2.3 Header style information: the `STYLE` element

```
<!ELEMENT STYLE - - %StyleSheet
<!ATTLIST STYLE
  %I18n;
  type %ContentType; #REQUIRED -- content type of style language --
  media %MediaDesc; #IMPLIED -- designed for use with these media --
  title %Text; #IMPLIED -- advisory title --
  >
```

Start tag: *required*, **End tag:** *required*

Attribute definitions

`type` = *content-type* [p.53] [CI] [p.49]

This attribute specifies the style sheet language of the element's contents and overrides the default style sheet language. The style sheet language is specified as a content type (e.g., "text/css"). Authors must supply a value for this attribute; there is no default value for this attribute.

`media` = *media-descriptors* [p.56] [CI] [p.49]

This attribute specifies the intended destination medium for style information. It may be a single media descriptor or a comma-separated list. The default value for this attribute is "screen".

Attributes defined elsewhere

- `lang` (language information [p.79]), `dir` (text direction [p.82])
- `title` (element title [p.63])

The `STYLE` element allows authors to put style sheet rules in the head of the document. HTML permits any number of `STYLE` elements in the `HEAD` section of a document.

User agents that don't support style sheets, or don't support the specific style sheet language used by a `STYLE` element, must hide the contents of the `STYLE` element. It is an error to render the content as part of the document's text. Some style sheet languages support syntax for hiding the content [p.193] from non-conforming user agents.

The syntax of style data [p.57] depends on the style sheet language.

Some style sheet implementations may allow a wider variety of rules in the `STYLE` element than in the `style` attribute. For example, with CSS, rules may be declared within a `STYLE` element for:

- All instances of a specific HTML element (e.g., all `P` elements, all `H1` elements, etc.)

- All instances of an HTML element belonging to a specific class (i.e., whose `class` attribute is set to some value).
- Single instances of an HTML element (i.e., whose `id` attribute is set to some value).

Rules for style rule precedences and inheritance depend on the style sheet language.

The following CSS `STYLE` declaration puts a border around every `H1` element in the document and centers it on the page.

```
<HEAD>
<STYLE type="text/css">
H1 {border-width: 1; border: solid; text-align: center}
</STYLE>
</HEAD>
```

To specify that this style information should only apply to `H1` elements of a specific class, we modify it as follows:

```
<HEAD>
<STYLE type="text/css">
H1.myclass {border-width: 1; border: solid; text-align: center}
</STYLE>
</HEAD>
<BODY>
<H1 class="myclass"> This H1 is affected by our style </H1>
<H1> This one is not affected by our style </H1>
</BODY>
```

Finally, to limit the scope of the style information to a single instance of `H1`, set the `id` attribute:

```
<HEAD>
<STYLE type="text/css">
#myid {border-width: 1; border: solid; text-align: center}
</STYLE>
</HEAD>
<BODY>
<H1 class="myclass"> This H1 is not affected </H1>
<H1 id="myid"> This H1 is affected by style </H1>
<H1> This H1 is not affected </H1>
</BODY>
```

Although style information may be set for almost every HTML element, two elements, `DIV` and `SPAN`, are particularly useful in that they do not impose any presentation semantics (besides block-level vs. inline [p.73]). When combined with style sheets, these elements allow users to extend HTML indefinitely, particularly when used with the `class` and `id` attributes.

In the following example, we use the `SPAN` element to set the font style of the first few words of a paragraph to small caps.

```

<HEAD>
<STYLE type="text/css">
SPAN.sc-ex { font-variant: small-caps }
</STYLE>
</HEAD>
</BODY>
<P><SPAN class="sc-ex">The first</SPAN> few words of
this paragraph are in small-caps.
</BODY>

```

In the following example, we use `DIV` and the `class` attribute to set the text justification for a series of paragraphs that make up the abstract section of a scientific article. This style information could be reused for other abstract sections by setting the `class` attribute elsewhere in the document.

```

<HEAD>
<STYLE type="text/css">
DIV.Abstract { text-align: justify }
</STYLE>
</HEAD>
<BODY>
<DIV class="Abstract">
<P>The Chieftain product range is our market winner for
the coming year. This report sets out how to position
Chieftain against competing products.
<P>Chieftain replaces the Commander range, which will
remain on the price list until further notice.
</DIV>
</BODY>

```

14.2.4 Media types

HTML allows authors to design documents that take advantage of the characteristics of the media where the document is to be rendered (e.g., graphical displays, television screens, handheld devices, speech-based browsers, braille-based tactile devices, etc.). By specifying the `media` attribute, authors allow user agents to load and apply style sheets selectively. Please consult the list of recognized media descriptors [p.56].

The following sample declarations apply to `H1` elements. When projected in a business meeting, all instances will be blue. When printed, all instances will be centered.

```

<HEAD>
<STYLE type="text/css" media="projection">
H1 { color: blue }
</STYLE>
<STYLE type="text/css" media="print">
H1 { text-align: center }
</STYLE>

```

This example adds sound effects to anchors for use in speech output:

```

<STYLE type="text/css" media="aural">
A { cue-before: uri(bell.aiff); cue-after: uri(dong.wav) }
</STYLE>
</HEAD>

```

Media control is particularly interesting when applied to external style sheets since user agents can save time by retrieving from the network only those style sheets that apply to the current device. For instance, speech-based browsers can avoid downloading style sheets designed for visual rendering. See the section on media-dependent cascades [p.192] for more information.

14.3 External style sheets

Authors may separate style sheets from HTML documents. This offers several benefits:

- Authors and Web site managers may share style sheets across a number of documents (and sites).
- Authors may change the style sheet without requiring modifications to the document.
- User agents may load style sheets selectively (based on media descriptions).

14.3.1 Preferred and alternate style sheets

HTML allows authors to associate any number of external style sheets with a document. The style sheet language defines how multiple external style sheets interact (for example, the CSS "cascade" rules).

Authors may specify a number of mutually exclusive style sheets called *alternate* style sheets. Users may select their favorite among these depending on their preferences. For instance, an author may specify one style sheet designed for small screens and another for users with weak vision (e.g., large fonts). User agents should allow users to select from alternate style sheets.

The author may specify that one of the alternates is a *preferred* style sheet. User agents should apply the author's preferred style sheet unless the user has selected a different alternate.

Authors may group several alternate style sheets (including the author's preferred style sheets) under a single *style name*. When a user selects a named style, the user agent must apply all style sheets with that name. User agents must not apply alternate style sheets with a different style name. The section on specifying external style sheets [p.191] explains how to name a group of style sheets.

Authors may also specify *persistent* style sheets that user agents must apply in addition to any alternate style sheet.

User agents must respect media descriptors [p.56] when applying any style sheet. User agents should also allow users to disable the author's style sheets entirely, in which case the user agent must not apply any persistent or alternate style sheets.

14.3.2 Specifying external style sheets

Authors specify external style sheets with the following attributes of the `LINK` element:

- Set the value of `href` to the location of the style sheet file. The value of `href` is a URI [p.51].
- Set the value of the `type` attribute to indicate the language of the linked (style sheet) resource. This allows the user agent to avoid downloading a style sheet for an unsupported style sheet language.
- Specify that the style sheet is persistent, preferred, or alternate:
 - To make a style sheet persistent, set the `rel` attribute to "stylesheet" and don't set the `title` attribute.
 - To make a style sheet preferred, set the `rel` attribute to "stylesheet" and name the style sheet with the `title` attribute.
 - To specify an alternate style sheet, set the `rel` attribute to "alternate stylesheet" and name the style sheet with the `title` attribute.

User agents should provide a means for users to view and pick from the list of alternate styles. The value of the `title` attribute is recommended as the name of each choice.

In this example, we first specify a persistent style sheet located in the file `mystyle.css`:

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

Setting the `title` attribute makes this the author's preferred style sheet:

```
<LINK href="mystyle.css" title="compact" rel="stylesheet" type="text/css" >
```

Adding the keyword "alternate" to the `rel` attribute makes it an alternate style sheet:

```
<LINK href="mystyle.css" title="Medium" rel="alternate stylesheet" type="text/css" >
```

For more information on external style sheets, please consult the section on links and external style sheets. [p.155]

Authors may also use the `META` element to set the document's preferred style sheet. For example, to set the preferred style sheet to "compact" (see the preceding example), authors may include the following line in the `HEAD`:

```
<META http-equiv="Default-Style" content="compact" >
```

The preferred style sheet may also be specified with HTTP headers. The above `META` declaration is equivalent to the HTTP header:

```
Default-Style: "compact"
```

If two or more `META` declarations or HTTP headers specify the preferred style sheet, the last one takes precedence. HTTP headers are considered to occur earlier than the document `HEAD` for this purpose.

If two or more `LINK` elements specify a preferred style sheet, the first one takes precedence.

Preferred style sheets specified with `META` or HTTP headers have precedence over those specified with the `LINK` element.

14.4 Cascading style sheets

Cascading style sheet languages such as CSS allow style information from several sources to be blended together. However, not all style sheet languages support cascading. To define a cascade, authors specify a sequence of `LINK` and/or `STYLE` elements. The style information is cascaded in the order the elements appear in the `HEAD`.

Note. *This specification does not specify how style sheets from different style languages cascade. Authors should avoid mixing style sheet languages.*

In the following example, we specify two alternate style sheets named "compact". If the user selects the "compact" style, the user agent must apply both external style sheets, as well as the persistent "common.css" style sheet. If the user selects the "big print" style, only the alternate style sheet "bigprint.css" and the persistent "common.css" will be applied.

```
<LINK rel="alternate stylesheet" title="compact" href="small-base.css" type="text/css" >
<LINK rel="alternate stylesheet" title="compact" href="small-extras.css" type="text/css" >
<LINK rel="alternate stylesheet" title="big print" href="bigprint.css" type="text/css" >
<LINK rel="stylesheet" href="common.css" type="text/css" >
```

Here is a cascade example that involves both the `LINK` and `STYLE` elements.

```
<LINK rel="stylesheet" href="corporate.css" type="text/css" >
<LINK rel="stylesheet" href="techreport.css" type="text/css" >
<STYLE type="text/css" >
  p.special { color: rgb(230, 100, 180) }
</STYLE>
```

14.4.1 Media-dependent cascades

A cascade may include style sheets applicable to different media. Both `LINK` and `STYLE` may be used with the `media` attribute. The user agent is then responsible for filtering out those style sheets that do not apply to the current medium.

In the following example, we define a cascade where the "corporate" style sheet is provided in several versions: one suited to printing, one for screen use and one for speech-based browsers (useful, say, when reading email in the car). The "techreport" stylesheet applies to all media. The color rule defined by the `STYLE` element is used for print and screen but not for aural rendering.

```
<LINK rel="stylesheet" media="aural" href="corporate-aural.css" type="text/css">
<LINK rel="stylesheet" media="screen" href="corporate-screen.css" type="text/css">
<LINK rel="stylesheet" media="print" href="corporate-print.css" type="text/css">
<LINK rel="stylesheet" href="techreport.css" type="text/css">
<STYLE media="screen, print" type="text/css">
  p.special { color: rgb(230, 100, 180) }
</STYLE>
```

14.4.2 Inheritance and cascading

When the user agent wants to render a document, it needs to find values for style properties, e.g. the font family, font style, size, line height, text color and so on. The exact mechanism depends on the style sheet language, but the following description is generally applicable:

The cascading mechanism is used when a number of style rules all apply directly to an element. The mechanism allows the user agent to sort the rules by specificity, to determine which rule to apply. If no rule can be found, the next step depends on whether the style property can be inherited or not. Not all properties can be inherited. For these properties the style sheet language provides default values for use when there are no explicit rules for a particular element.

If the property can be inherited, the user agent examines the immediately enclosing element to see if a rule applies to that. This process continues until an applicable rule is found. This mechanism allows style sheets to be specified compactly. For instance, authors may specify the font family for all elements within the `BODY` by a single rule that applies to the `BODY` element.

14.5 Hiding style data from user agents

Some style sheet languages support syntax intended to allow authors to hide the content of `STYLE` elements from non-conforming user agents.

This example illustrates for CSS how to comment out the content of `STYLE` elements to ensure that older, non-conforming user agents will not render them as text.

```
<STYLE type="text/css" >
<!--
  H1 { color: red }
  P  { color: blue }
-->
</STYLE>
```

14.6 Linking to style sheets with HTTP headers

This section only applies to user agents conforming to versions of HTTP that define a `Link` header field. Note that HTTP 1.1 as defined by [RFC2616] [p. 354] does not include a `Link` header field (refer to section 19.6.3).

Web server managers may find it convenient to configure a server so that a style sheet will be applied to a group of pages. The `HTTP` `Link` header has the same effect as a `LINK` element with the same attributes and values. Multiple `Link` headers correspond to multiple `LINK` elements occurring in the same order. For instance,

```
Link: <http://www.acme.com/corporate.css>; REL=stylesheet
```

corresponds to:

```
<LINK rel="stylesheet" href="http://www.acme.com/corporate.css">
```

It is possible to specify several alternate styles using multiple `Link` headers, and then use the `rel` attribute to determine the default style.

In the following example, "compact" is applied by default since it omits the "alternate" keyword for the `rel` attribute.

```
Link: <compact.css>; rel="stylesheet"; title="compact"
Link: <bigprint.css>; rel="alternate stylesheet"; title="big print"
```

This should also work when HTML documents are sent by email. Some email agents can alter the ordering of [RFC822] [p.356] headers. To protect against this affecting the cascading order for style sheets specified by `Link` headers, authors can use header concatenation to merge several instances of the same header field. The quote marks are only needed when the attribute values include whitespace. Use SGML entities to reference characters that are otherwise not permitted within HTTP or email headers, or that are likely to be affected by transit through gateways.

`LINK` and `META` elements implied by HTTP headers are defined as occurring before any explicit `LINK` and `META` elements in the document's `HEAD`.

15 Alignment, font styles, and horizontal rules

Contents

1. Formatting	195
1. Background color	195
2. Alignment	195
3. Floating objects	197
• Float an object	197
• Float text around an object	198
2. Fonts	199
1. Font style elements: the <code>TT</code> , <code>I</code> , <code>B</code> , <code>BIG</code> , <code>SMALL</code> , <code>STRIKE</code> , <code>S</code> , and <code>U</code> elements	199
2. Font modifier elements: <code>FONT</code> and <code>BASEFONT</code>	200
3. Rules: the <code>HR</code> element	202

This section of the specification discusses some HTML elements and attributes that may be used for visual formatting of elements. Many of them are deprecated [p.38].

15.1 Formatting

15.1.1 Background color

Attribute definitions

`bgcolor = color` [p.51] [CI] [p.49]

Deprecated. [p.38] This attribute sets the background color for the document body or table cells.

This attribute sets the background color of the canvas for the document body (the `BODY` element) or for tables (the `TABLE`, `TR`, `TH`, and `TD` elements). Additional attributes for specifying text color can be used with the `BODY` element.

This attribute has been deprecated [p.38] in favor of style sheets for specifying background color information.

15.1.2 Alignment

It is possible to align block elements (tables, images, objects, paragraphs, etc.) on the canvas with the `align` attribute. Although this attribute may be set for many HTML elements, its range of possible values sometimes differs from element to element. Here we only discuss the meaning of the `align` attribute for text.

Attribute definitions

`align = left|center|right|justify` [CI] [p.49]

Deprecated. [p.38] This attribute specifies the horizontal alignment of its element with respect to the surrounding context. Possible values:

- `left`: text lines are rendered flush left.
- `center`: text lines are centered.
- `right`: text lines are rendered flush right.
- `justify`: text lines are justified to both margins.

The default depends on the base text direction. For left to right text, the default is `align=left`, while for right to left text, the default is `align=right`.

DEPRECATED EXAMPLE:

This example centers a heading on the canvas.

```
<H1 align="center"> How to Carve Wood </H1>
```

Using CSS, for example, you could achieve the same effect as follows:

```
<HEAD>
<TITLE>How to Carve Wood</TITLE>
<STYLE type="text/css">
H1 { text-align: center }
</STYLE>
<BODY>
<H1> How to Carve Wood </H1>
```

Note that this would center all `H1` declarations. You could reduce the scope of the style by setting the `class` attribute on the element:

```
<HEAD>
<TITLE>How to Carve Wood</TITLE>
<STYLE type="text/css">
H1.wood {text-align: center }
</STYLE>
<BODY>
<H1 class="wood"> How to Carve Wood </H1>
```

DEPRECATED EXAMPLE:

Similarly, to right align a paragraph on the canvas with HTML's `align` attribute you could have:

```
<P align="right">...Lots of paragraph text...
```

which, with CSS, would be:

```
<HEAD>
<TITLE>How to Carve Wood</TITLE>
<STYLE type="text/css">
P.mypar {text-align: right }
</STYLE>
<BODY>
<P class="mypar">...Lots of paragraph text...
```

DEPRECATED EXAMPLE:

To right align a series of paragraphs, group them with the `DIV` element:

```
<DIV align="right">
<P>...text in first paragraph...
<P>...text in second paragraph...
<P>...text in third paragraph...
</DIV>
```

With CSS, the `text-align` property is inherited from the parent element, you can therefore use:

```
<HEAD>
<TITLE>How to Carve Wood</TITLE>
<STYLE type="text/css">
DIV.mypars {text-align: right}
</STYLE>
<BODY>
<DIV class="mypars">
<P>...text in first paragraph...
<P>...text in second paragraph...
<P>...text in third paragraph...
</DIV>
```

To center the entire document with CSS:

```
<HEAD>
<TITLE>How to Carve Wood</TITLE>
<STYLE type="text/css">
BODY {text-align: center}
</STYLE>
<BODY>
...the body is centered...
</BODY>
```

The `CENTER` element is exactly equivalent to specifying the `DIV` element with the `align` attribute set to "center". **The `CENTER` element is deprecated [p.38]**.

15.1.3 Floating objects

Images and objects may appear directly "in-line" or may be floated to one side of the page, temporarily altering the margins of text that may flow on either side of the object.

Float an object

The `align` attribute for objects, images, tables, frames, etc., causes the object to float to the left or right margin. Floating objects generally begin a new line. This attribute takes the following values:

- `left`: Floats the object to the current left margin. Subsequent text flows along the image's right side.
- `right`: Floats the object to the current right margin. Subsequent text flows along the image's left side.

DEPRECATED EXAMPLE:

The following example shows how to float an `IMG` element to the current left margin of the canvas.

```
<IMG align="left" src="http://foo.com/animimage.gif" alt="my boat">
```

Some alignment attributes also permit the "center" value, which does not cause floating, but centers the object within the current margins. However, for `P` and `DIV`, the value "center" causes the contents of the element to be centered.

Float text around an object

Another attribute, defined for the `BR` element, controls text flow around floating objects.

Attribute definitions

```
clear = none | left | right | all [CI] [p.49]
```

Deprecated. [p.38] Specifies where the next line should appear in a visual browser after the line break caused by this element. This attribute takes into account floating objects (images, tables, etc.). Possible values:

- `none`: The next line will begin normally. This is the default value.
- `left`: The next line will begin at nearest line below any floating objects on the left-hand margin.
- `right`: The next line will begin at nearest line below any floating objects on the right-hand margin.
- `all`: The next line will begin at nearest line below any floating objects on either margin.

Consider the following visual scenario, where text flows to the right of an image until a line is broken by a `BR`:

```
*****
|-----|
| image |--<BR>
|-----|
*****
```

If the `clear` attribute is set to `none`, the line following `BR` will begin immediately below it at the right margin of the image:

```
*****
|-----|
| image |--<BR>
|-----|
*****
```

DEPRECATED EXAMPLE:

If the `clear` attribute is set to `left` or `all`, the next line will appear as follows:

DEPRECATED EXAMPLE:

To right align a series of paragraphs, group them with the `DIV` element:

```
<DIV align="right">
<P>...text in first paragraph...
<P>...text in second paragraph...
<P>...text in third paragraph...
</DIV>
```

With CSS, the `text-align` property is inherited from the parent element, you can therefore use:

```
<HEAD>
<TITLE>How to Carve Wood</TITLE>
<STYLE type="text/css">
DIV.mypars {text-align: right}
</STYLE>
<BODY>
<DIV class="mypars">
<P>...text in first paragraph...
<P>...text in second paragraph...
<P>...text in third paragraph...
</DIV>
```

To center the entire document with CSS:

```
<HEAD>
<TITLE>How to Carve Wood</TITLE>
<STYLE type="text/css">
BODY {text-align: center}
</STYLE>
<BODY>
...the body is centered...
</BODY>
```

The `CENTER` element is exactly equivalent to specifying the `DIV` element with the `align` attribute set to "center". **The `CENTER` element is deprecated [p.38]**.

15.1.3 Floating objects

Images and objects may appear directly "in-line" or may be floated to one side of the page, temporarily altering the margins of text that may flow on either side of the object.

Float an object

The `align` attribute for objects, images, tables, frames, etc., causes the object to float to the left or right margin. Floating objects generally begin a new line. This attribute takes the following values:

- `left`: Floats the object to the current left margin. Subsequent text flows along the image's right side.
- `right`: Floats the object to the current right margin. Subsequent text flows along the image's left side.

```

*****
|-----|
| image |  --<BR clear="left">
|-----|
*****
|-----|

```

Using style sheets, you could specify that all line breaks should behave this way for objects (images, tables, etc.) floating against the left margin. With CSS, you could achieve this as follows:

```

<STYLE type="text/css">
BR { clear: left }
</STYLE>

```

To specify this behavior for a specific instance of the BR element, you could combine style information and the id attribute:

```

<HEAD>
...
<STYLE type="text/css">
BR#mybr { clear: left }
</STYLE>
</HEAD>
<BODY>
<P>...
*****
| table |  --<BR id="mybr">
|-----|
*****
...
</BODY>

```

15.2 Fonts

The following HTML elements specify font information. Although they are not all deprecated [p.38], their use is discouraged in favor of style sheets.

15.2.1 Font style elements: the TT, I, B, BIG, SMALL, STRIKE, S, and U elements

```

<!ENTITY % fontStyle
"TT | I | B | BIG | SMALL">
<!ELEMENT (%fontstyle;|phrase;) - - (%inline;)*>
<!ATTLIST (%fontstyle;|phrase;)
  %attrs;
  -- %coreattrs, %i18n, %events --
>

```

Start tag: **required**, End tag: **required**

Attributes defined elsewhere

- id, class (document-wide identifiers [p.71])
- lang (language information [p.79]), dir (text direction [p.82])
- title (element title [p.63])
- style (inline style information [p.186])
- onclick, ondblclick, onmousedown, onmouseover, onmousemove, onmouseup, onmouseout, onkeypress, onkeydown, onkeyup (intrinsic events [p.254])

Rendering of font style elements depends on the user agent. The following is an informative description only.

TT: Renders as teletype or monospaced text.

I: Renders as italic text style.

B: Renders as bold text style.

BIG: Renders text in a "large" font.

SMALL: Renders text in a "small" font.

STRIKE and **S**: **Deprecated**. [p.38] Render strike-through style text.

U: **Deprecated**. [p.38] Renders underlined text.

The following sentence shows several types of text:

```

<P><b>bold</b>,
<i>italic</i>, <b><i>bold italic</i></b>, <tt>teletype text</tt>, and
<big>big</big> and <small>small</small> text.

```

These words might be rendered as follows:

bold, *italic*, **bold italic**, teletype text, and big and small text.

It is possible to achieve a much richer variety of font effects using style sheets. To specify blue, italic text in a paragraph with CSS:

```

<HEAD>
<STYLE type="text/css">
P#mypar {font-style: italic; color: blue}
</STYLE>
</HEAD>
<P id="mypar">...Lots of blue italic text...

```

Font style elements must be properly nested. Rendering of nested font style elements depends on the user agent.

15.2.2 Font modifier elements: FONT and BASEFONT

FONT and **BASEFONT** are deprecated [p.38].

See the Transitional DTD [p.283] for the formal definition.

Attribute definitions

`size = cdata` [p.50] [CN] [p.49]

Deprecated. [p.38] This attribute sets the size of the font. Possible values:

- An integer between 1 and 7. This sets the font to some fixed size, whose rendering depends on the user agent. Not all user agents may render all seven sizes.
- A relative increase in font size. The value "+1" means one size larger. The value ".3" means three sizes smaller. All sizes belong to the scale of 1 to 7.

`color = color` [p.51] [CI] [p.49]

Deprecated. [p.38] This attribute sets the text color.

`face = cdata` [p.50] [CI] [p.49]

Deprecated. [p.38] This attribute defines a comma-separated list of font names the user agent should search for in order of preference.

Attributes defined elsewhere

- `id`, `class` (document-wide identifiers [p.71])
- `lang` (language information [p.79]), `dir` (text direction [p.82])
- `title` (element title [p.63])
- `style` (inline style information [p.186])

The `FONT` element changes the font size and color for text in its contents.

The `BASEFONT` element sets the base font size (using the `size` attribute). Font size changes achieved with `FONT` are relative to the base font size set by `BASEFONT`. If `BASEFONT` is not used, the default base font size is 3.

DEPRECATED EXAMPLE:

The following example will show the difference between the seven font sizes available with `FONT`:

```
<p><font size=1>size=1</font>
<font size=2>size=2</font>
<font size=3>size=3</font>
<font size=4>size=4</font>
<font size=5>size=5</font>
<font size=6>size=6</font>
<font size=7>size=7</font>
```

This might be rendered as:

```
size=1 size=2 size=3 size=4 size=5 size=6 size=7
```

The following shows an example of the effect of relative font sizes using a base font size of 3:

```
size=-4 size=-3 size=-2 size=-1 size=+1 size=+2 size=+3 size=+4
```

The base font size does not apply to headings, except where these are modified using the `FONT` element with a relative font size change.

15.3 Rules: the `HR` element

```
<!ELEMENT HR - O EMPTY -- horizontal rule -->
<!ATTLIST HR
  %attrs;
  -- %coreattrs, %ilbn, %events --
>
```

Start tag: required, End tag: forbidden

Attribute definitions

`align = left|center|right` [CI] [p.49]

Deprecated. [p.38] This attribute specifies the horizontal alignment of the rule with respect to the surrounding context. Possible values:

- `left`: the rule is rendered flush left.
- `center`: the rule is centered.
- `right`: the rule is rendered flush right.

The default is `align=center`.

`noshade` [CI] [p.49]

Deprecated. [p.38] When set, this boolean attribute requests that the user agent render the rule in a solid color rather than as the traditional two-color "groove".

`size = pixels` [p.52] [CI] [p.49]

Deprecated. [p.38] This attribute specifies the height of the rule. The default value for this attribute depends on the user agent.

`width = length` [p.52] [CI] [p.49]

Deprecated. [p.38] This attribute specifies the width of the rule. The default width is 100%, i.e., the rule extends across the entire canvas.

Attributes defined elsewhere

- `id`, `class` (document-wide identifiers [p.71])
- `lang` (language information [p.79]), `dir` (text direction [p.82])
- `title` (element title [p.63])
- `style` (inline style information [p.186])
- `onclick`, `ondblclick`, `onmousedown`, `onmouseup`, `onmouseover`, `onmousemove`, `onmouseout`, `onkeypress`, `onkeydown`, `onkeyup` (intrinsic events [p.254])

The `HR` element causes a horizontal rule to be rendered by visual user agents.

The amount of vertical space inserted between a rule and the content that surrounds it depends on the user agent.

DEPRECATED EXAMPLE:

This example centers the rules, sizing them to half the available width between the margins. The top rule has the default thickness while the bottom two are set to 5 pixels. The bottom rule should be rendered in a solid color without shading:

```
<HR width="50%" align="center" >  
<HR size="5" width="50%" align="center" >  
<HR noshade size="5" width="50%" align="center" >
```

These rules might be rendered as follows:



16 Frames

Contents

- 1. Introduction to frames 205
- 2. Layout of frames 206
 - 1. The FRAMESET element 206
 - Rows and columns 207
 - Nested frame sets 208
 - Sharing data among frames 208
 - 2. The FRAME element 209
 - Setting the initial contents of a frame 210
 - Visual rendering of a frame 212
- 3. Specifying target frame information 212
 - 1. Setting the default target for links 213
 - 2. Target semantics 214
- 4. Alternate content 214
 - 1. The NOFRAMES element 214
 - 2. Long descriptions of frames 215
- 5. Inline frames: the IFRAME element 217

16.1 Introduction to frames

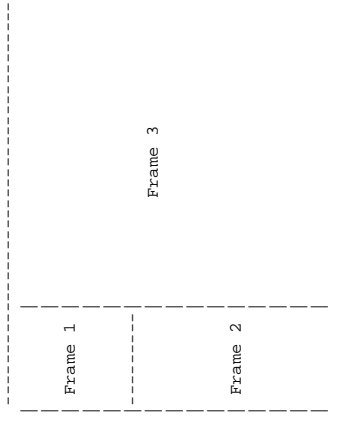
HTML frames allow authors to present documents in multiple views, which may be independent windows or subwindows. Multiple views offer designers a way to keep certain information visible, while other views are scrolled or replaced. For example, within the same window, one frame might display a static banner, a second a navigation menu, and a third the main document that can be scrolled through or replaced by navigating in the second frame.

Here is a simple frame document:

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01 Frameset//EN"
"http://www.w3.org/TR/html4/frameset.dtd">
<HTML>
<HEAD>
<TITLE>A simple frameset document</TITLE>
</HEAD>
<FRAMESET COLS="20%, 80%">
<FRAMESET ROWS="100, 200">
  <FRAME SRC="contents_of_frame1.html">
  <FRAME SRC="contents_of_frame2.gif">
</FRAMESET>
<FRAME SRC="contents_of_frame3.html">
<NOFRAMES>
<P>This frameset document contains:
</P>
</UI>
<LI><A href="contents_of_frame1.html">Some neat contents</A>
<LI><IMG SRC="contents_of_frame2.gif" alt="A neat image">
```

```
<LI><A href="contents_of_frame3.html">Some other neat contents</A>
</UI>
</NOFRAMES>
</FRAMESET>
</HTML>
```

that might create a frame layout something like this:



If the user agent can't display frames or is configured not to, it will render the contents of the NOFRAMES element.

16.2 Layout of frames

An HTML document that describes frame layout (called a *frameset document*) has a different makeup than an HTML document without frames. A standard document has one HEAD section and one BODY. A frameset document has a HEAD, and a FRAMESET in place of the BODY.

The FRAMESET section of a document specifies the layout of views in the main user agent window. In addition, the FRAMESET section can contain a NOFRAMES element to provide alternate content [p.214] for user agents that do not support frames or are configured not to display frames.

Elements that might normally be placed in the BODY element must not appear before the first FRAMESET element or the FRAMESET will be ignored.

16.2.1 The FRAMESET element

```
<!-- HTML Frameset; [
<!ELEMENT FRAMESET -- ((FRAMESET|FRAME)+ & NOFRAMES?) -- window subdivision-->
<!ATTLIST FRAMESET
%coreattrs;
rows %MultiLengths; #IMPLIED -- list of lengths,
-- id, class, style, title --
-- default: 100% (1 row) --
cols %MultiLengths; #IMPLIED -- list of lengths,
-- default: 100% (1 col) --
```

```

onload      %Script;      #IMPLIED -- all the frames have been loaded --
onunload    %Script;      #IMPLIED -- all the frames have been removed --
>
]]>

```

Attribute definitions

`rows` = *multi-length-list* [p.52] [CN] [p.49]

This attribute specifies the layout of horizontal frames. It is a comma-separated list of pixels, percentages, and relative lengths. The default value is 100%, meaning one row.

`cols` = *multi-length-list* [p.52] [CN] [p.49]

This attribute specifies the layout of vertical frames. It is a comma-separated list of pixels, percentages, and relative lengths. The default value is 100%, meaning one column.

Attributes defined elsewhere

- `id`, `class` (document-wide identifiers [p.71])
- `title` (element title [p.63])
- `style` (inline style information [p.186])
- `onload`, `onunload` (intrinsic events [p.254])

The `FRAMESET` element specifies the layout of the main user window in terms of rectangular subspaces.

Rows and columns

Setting the `rows` attribute defines the number of horizontal subspaces in a frameset. Setting the `cols` attribute defines the number of vertical subspaces. Both attributes may be set simultaneously to create a grid.

If the `rows` attribute is not set, each column extends the entire length of the page. If the `cols` attribute is not set, each row extends the entire width of the page. If neither attribute is set, the frame takes up exactly the size of the page.

Frames are created left-to-right for columns and top-to-bottom for rows. When both attributes are specified, views are created left-to-right in the top row, left-to-right in the second row, etc.

The first example divides the screen vertically in two (i.e., creates a top half and a bottom half).

```

<FRAMESET rows="50%, 50%">
...the rest of the definition...
</FRAMESET>

```

The next example creates three columns: the second has a fixed width of 250 pixels (useful, for example, to hold an image with a known size). The first receives 25% of the remaining space and the third 75% of the remaining space.

```

<FRAMESET cols="1*, 250, 3*">
...the rest of the definition...
</FRAMESET>

```

The next example creates a 2x3 grid of subspaces.

```

<FRAMESET rows="30%, 70%" cols="33%, 34%, 33%">
...the rest of the definition...
</FRAMESET>

```

For the next example, suppose the browser window is currently 1000 pixels high. The first view is allotted 30% of the total height (300 pixels). The second view is specified to be exactly 400 pixels high. This leaves 300 pixels to be divided between the other two frames. The fourth frame's height is specified as "2*", so it is twice as high as the third frame, whose height is only "*" (equivalent to 1*). Therefore the third frame will be 100 pixels high and the fourth will be 200 pixels high.

```

<FRAMESET rows="30%, 400, *, 2*">
...the rest of the definition...
</FRAMESET>

```

Absolute lengths that do not sum to 100% of the real available space should be adjusted by the user agent. When underspecified, remaining space should be allotted proportionally to each view. When overspecified, each view should be reduced according to its specified proportion of the total space.

Nested frame sets

Framesets may be nested to any level.

In the following example, the outer `FRAMESET` divides the available space into three equal columns. The inner `FRAMESET` then divides the second area into two rows of unequal height.

```

<FRAMESET cols="33%, 33%, 34%">
...contents of first frame...
<FRAMESET rows="40%, 50%">
...contents of second frame, first row...
...contents of second frame, second row...
</FRAMESET>
...contents of third frame...
</FRAMESET>

```

Sharing data among frames

Authors may share data among several frames by including this data via an `OBJECT` element. Authors should include the `OBJECT` element in the `HEAD` element of a frameset document and name it with the `id` attribute. Any document that is the contents of a frame in the frameset may refer to this identifier.

The following example illustrates how a script might refer to an `OBJECT` element defined for an entire frameset:


```

<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01 Frameset//EN"
"http://www.w3.org/TR/html4/frameset.dtd">
<HTML>
<HEAD>
<TITLE>This is a frameset with OBJECT in the HEAD</TITLE>
<!-- This OBJECT is not rendered! -->
<OBJECT id="myobject" data="data.bar"></OBJECT>
</HEAD>
<FRAMESET>
  <FRAME src="bianca.html" name="bianca" >
</FRAMESET>
</HTML>

<!-- In bianca.html -->
<HTML>
<HEAD>
<TITLE>Bianca's page</TITLE>
</HEAD>
<BODY>
  ...the beginning of the document...
<P>
<SCRIPT type="text/javascript">
parent.myobject.myproperty
</SCRIPT>
  ...the rest of the document...
</BODY>
</HTML>

```

16.2.2 The FRAME element

```

<{ %HTML.Frameset; {
<!-- reserved frame names start with "_" otherwise starts with letter -->
<ELEMENT FRAME - O EMPTY
<!ATTLIST FRAME
  %coreattrs: %URI; #IMPLIED
  longdesc #IMPLIED
  name CDATA #IMPLIED
  src %URI; #IMPLIED
  frameborder (1|0) 1
  marginwidth %Pixels; #IMPLIED
  marginheight %Pixels; #IMPLIED
  noresize (noresize) #IMPLIED
  scrolling (yes|no|auto) auto
  >
}]>

```

Attribute definitions

`name = cdata` [p.50] [CI] [p.49]

This attribute assigns a name to the current frame. This name may be used as the target of subsequent links.

`longdesc = uri` [p.51] [CT] [p.49]

This attribute specifies a link to a long description of the frame. This description should supplement the short description provided using the `title` attribute, and

may be particularly useful for non-visual user agents.

`src = uri` [p.51] [CT] [p.49]

This attribute specifies the location of the initial contents to be contained in the frame.

`noresize` [CI] [p.49]

When present, this boolean attribute tells the user agent that the frame window must not be resizable.

`scrolling = auto|yes|no` [CI] [p.49]

This attribute specifies scroll information for the frame window. Possible values

- `auto`: This value tells the user agent to provide scrolling devices for the frame window when necessary. This is the default value.

- `yes`: This value tells the user agent to always provide scrolling devices for the frame window.

- `no`: This value tells the user agent not to provide scrolling devices for the frame window.

`frameborder = 1|0` [CN] [p.49]

This attribute provides the user agent with information about the frame border.

Possible values:

- `1`: This value tells the user agent to draw a separator between this frame and every adjoining frame. This is the default value.

- `0`: This value tells the user agent not to draw a separator between this frame and every adjoining frame. Note that separators may be drawn next

to this frame nonetheless if specified by other frames.

`marginwidth = pixels` [p.52] [CN] [p.49]

This attribute specifies the amount of space to be left between the frame's

contents in its left and right margins. The value must be greater than zero (pixels). The default value depends on the user agent.

`marginheight = pixels` [p.52] [CN] [p.49]

This attribute specifies the amount of space to be left between the frame's contents in its top and bottom margins. The value must be greater than zero (pixels). The default value depends on the user agent.

Attributes defined elsewhere

- `id`, `class` (document-wide identifiers [p.71])
- `title` (element title [p.63])
- `style` (inline style information [p.186])

The `FRAME` element defines the contents and appearance of a single frame.

Setting the initial contents of a frame

The `src` attribute specifies the initial document the frame will contain.

The following example HTML document:


```

<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01 Frameset//EN"
"http://www.w3.org/TR/html4/frameset.dtd">
<HTML>
<HEAD>
<TITLE>A frameset document</TITLE>
</HEAD>
<FRAMESET rows="50%,50%">
  <FRAME name="fixed" src="init_fixed.html">
  <FRAME name="dynamic" src="init_dynamic.html">
</FRAMESET>
</HTML>

```

Then, in `init_dynamic.html`, we link to the frame named "dynamic".

```

<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN"
"http://www.w3.org/TR/html4/loose.dtd">
<HTML>
<HEAD>
<TITLE>A document with anchors with specific targets</TITLE>
</HEAD>
<BODY>
...beginning of the document...
<p>Now you may advance to
  <a href="slide2.html" target="dynamic">slide 2.</a>
...more document...
<p>You're doing great. Now on to
  <a href="slide3.html" target="dynamic">slide 3.</a>
</BODY>
</HTML>

```

Activating either link opens a new document in the frame named "dynamic" while the other frame, "fixed", maintains its initial contents.

Note. A frameset definition never changes, but the contents of one of its frames can. Once the initial contents of a frame change, the frameset definition no longer reflects the current state of its frames.

There is currently no way to encode the entire state of a frameset in a URI. Therefore, many user agents do not allow users to assign a bookmark to a frameset. Framesets may make navigation forward and backward through your user agent's history more difficult for users.

16.3.1 Setting the default target for links

When many links in the same document designate the same target, it is possible to specify the target once and dispense with the target attribute of each element. This is done by setting the target attribute of the BASE element.

We return to the previous example, this time factoring the target information by defining it in the BASE element and removing it from the A elements.

```

<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN"
"http://www.w3.org/TR/html4/loose.dtd">
<HTML>
<HEAD>

```

```

<TITLE>A document with BASE with a specific target</TITLE>
<BASE href="http://www.mycom.com/Slides" target="dynamic">
</HEAD>
<BODY>
...beginning of the document...
<p>Now you may advance to <a href="slide2.html">slide 2.</a>
...more document...
<p>You're doing great. Now on to
  <a href="slide3.html">slide 3.</a>
</BODY>
</HTML>

```

16.3.2 Target semantics

User agents should determine the target frame in which to load a linked resource according to the following precedences (highest priority to lowest):

1. If an element has its target attribute set to a known frame, when the element is activated (i.e., a link is followed or a form is processed), the resource designated by the element should be loaded into the target frame.
2. If an element does not have the target attribute set but the BASE element does, the BASE element's target attribute determines the frame.
3. If neither the element nor the BASE element refers to a target, the resource designated by the element should be loaded into the frame containing the element.
4. If any target attribute refers to an unknown frame F, the user agent should create a new window and frame, assign the name F to the frame, and load the resource designated by the element in the new frame.

User agents may provide users with a mechanism to override the target attribute.

16.4 Alternate content

Authors should supply alternate content for those user agents that do not support frames or are configured not to display frames.

16.4.1 The NOFRAMES element

```

<!-- HTML Frameset: [
<!-- ENTITY % noframes.content "(BODY) -(NOFRAMES)" -->
]]>
<!-- ENTITY % noframes.content "({flow:})" -->
<!-- ELEMENT NOFRAMES - - %noframes.content;
-- alternate content container for non frame-based rendering -->
<!-- LIST NOFRAMES
%attrs;
-- %coreattrs, %l18n, %events --
>

```

Attributes defined elsewhere

- `id`, `class` (document-wide identifiers [p.71])
- `lang` (language information [p.79]), `dir` (text direction [p.82])
- `title` (element title [p.63])
- `style` (inline style information [p.186])
- `onclick`, `ondblclick`, `onmousedown`, `onmouseup`, `onmouseover`, `onmousemove`, `onmouseout`, `onkeypress`, `onkeydown`, `onkeyup` (intrinsic events [p.254])

The `NOFRAMES` element specifies content that should be displayed only by user agents that do not support frames or are configured not to display frames. User agents that support frames must only display the contents of a `NOFRAMES` declaration when configured not to display frames. User agents that do not support frames must display the contents of `NOFRAMES` in any case.

The `NOFRAMES` element is part of both the transitional and frameset DTDs. [p.60] In a document that uses the frameset DTD, `NOFRAMES` may be used at the end of the `FRAMESET` section of the document.

For example:

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01 Frameset//EN"
"http://www.w3.org/TR/html4/frameset.dtd">
<HTML>
<HEAD>
<TITLE>A frameset document with NOFRAMES</TITLE>
</HEAD>
<FRAMESET cols="50%, 50%">
<FRAME src="main.html">
<FRAME src="table_of_contents.html">
<NOFRAMES>
<p>Here is the <A href="main-noframes.html">
non-frame based version of the document.</A>
</NOFRAMES>
</FRAMESET>
</HTML>
```

`NOFRAMES` may be used, for example, in a document that is the source of a frame and that uses the transitional DTD. This allows authors to explain the document's purpose in cases when it is viewed out of the frameset or with a user agent that doesn't support frames.

16.4.2 Long descriptions of frames

The `longdesc` attribute allows authors to make frame documents more accessible to people using non-visual user agents. This attribute designates a resource that provides a long description of the frame. Authors should note that long descriptions associated with frames are attached to the *frame*, not the frame's contents. Since the contents may vary over time, the initial long description is likely to become inappropriate for the frame's later contents. In particular, authors should not include

an image as the sole content of a frame.

The following frameset document describes two frames. The left frame contains a table of contents and the right frame initially contains an image of an ostrich:

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01 Frameset//EN"
"http://www.w3.org/TR/html4/frameset.dtd">
<HTML>
<HEAD>
<TITLE>A poorly-designed frameset document</TITLE>
</HEAD>
<FRAMESET cols="20%, 80%">
<FRAME src="table_of_contents.html">
<FRAME src="ostrich.gif" longdesc="ostrich-desc.html">
</FRAMESET>
</HTML>
```

Note that the image has been included in the frame independently of any HTML element, so the author has no means of specifying alternate text other than via the `longdesc` attribute. If the contents of the right frame change (e.g., the user selects a rattle-snake from the table of contents), users will have no textual access to the frame's new content.

Thus, authors should not put an image directly in a frame. Instead, the image should be specified in a separate HTML document, and therein annotated with the appropriate alternate text:

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01 Frameset//EN"
"http://www.w3.org/TR/html4/frameset.dtd">
<HTML>
<HEAD>
<TITLE>A well-designed frameset document</TITLE>
</HEAD>
<FRAMESET cols="20%, 80%">
<FRAME src="table_of_contents.html">
<FRAME src="ostrich-container.html">
</FRAMESET>
</HTML>
<!-- In ostrich-container.html: -->
<HTML>
<HEAD>
<TITLE>The fast and powerful ostrich</TITLE>
</HEAD>
<p>
<OBJECT data="ostrich.gif" type="image/gif">
These ostriches sure taste good!
</OBJECT>
</HTML>
```

16.5 Inline frames: the IFRAME element

```

<!ELEMENT IFRAME -- (%flow;)* -- inline subwindow -->
<!ATTLIST IFRAME
  %coreattrs;
  longdesc #IMPLIED -- id, class, style, title --
  %URI; #IMPLIED -- link to long description
  (complements title) --
  name #IMPLIED -- name of frame for targetting --
  src %URI; #IMPLIED -- source of frame content --
  1 -- request frame borders? --
  frameborder (1|0) #IMPLIED -- margin widths in pixels --
  marginwidth %Pixels; #IMPLIED -- margin height in pixels --
  marginheight %Pixels; #IMPLIED -- scrollbar or none --
  scrolling (yes|no|auto) auto -- vertical or horizontal alignment --
  align %Align; #IMPLIED -- frame height --
  height %Length; #IMPLIED -- frame width --
  width %Length; #IMPLIED -- frame width --
  >

```

Attribute definitions

`longdesc = uri` [p.51] [CT] [p.49]

This attribute specifies a link to a long description of the frame. This description should supplement the short description provided using the `title` attribute, and is particularly useful for non-visual user agents.

`name = cdata` [p.50] [C] [p.49]

This attribute assigns a name to the current frame. This name may be used as the target of subsequent links.

`width = length` [p.52] [CN] [p.49]

The width of the inline frame.

`height = length` [p.52] [CN] [p.49]

The height of the inline frame.

Attributes defined elsewhere

- `id, class` (document-wide identifiers [p.71])
- `title` (element title [p.63])
- `style` (inline style information [p.186])
- `name, src, frameborder, marginwidth, marginheight, scrolling` (frame controls and decoration [p.209])
- `align` (alignment [p.195])

The IFRAME element allows authors to insert a frame within a block of text. Inserting an inline frame within a section of text is much like inserting an object via the OBJECT element: they both allow you to insert an HTML document in the middle of another, they may both be aligned with surrounding text, etc.

The information to be inserted inline is designated by the `src` attribute of this element. The *contents* of the IFRAME element, on the other hand, should only be displayed by user agents that do not support frames or are configured not to display frames.

For user agents that support frames, the following example will place an inline frame surrounded by a border in the middle of the text.

```

<IFRAME src="foo.html" width="400" height="500"
  scrolling="auto" frameborder="1">
  [Your user agent does not support frames or is
  currently configured not to display frames.
  However, you may visit <A href="foo.html">the
  related document.</A>]
</IFRAME>

```

Inline frames may not be resized (and thus, they do not take the `noresize` attribute).

Note. HTML documents may also be embedded in other HTML documents with the OBJECT element. See the section on embedded documents [p.173] for details.

17 Forms

Contents

1. Introduction to forms	219
2. Controls	220
1. Control types	221
3. The FORM element	222
4. The INPUT element	224
1. Control types created with INPUT	226
2. Examples of forms containing INPUT controls	227
5. The BUTTON element	228
6. The SELECT, OPTGROUP, and OPTION elements	230
1. Pre-selected options	231
7. The TEXTAREA element	234
8. The ISINDEX element	236
9. Labels	237
1. The LABEL element	237
10. Adding structure to forms: the FIELDSET and LEGEND elements	239
11. Giving focus to an element	241
1. Tabbing navigation	241
2. Access keys	242
12. Disabled and read-only controls	243
1. Disabled controls	244
2. Read-only controls	244
13. Form submission	245
1. Form submission method	245
2. Successful controls	245
3. Processing form data	246
• Step one: Identify the successful controls	246
• Step two: Build a form data set	246
• Step three: Encode the form data set	246
• Step four: Submit the encoded form data set	247
4. Form content types	247
• application/x-www-form-urlencoded	247
• multipart/form-data	248

17.1 Introduction to forms

An HTML form is a section of a document containing normal content, markup, special elements called *controls* [p.220] (checkboxes, radio buttons, menus, etc.), and labels on those controls. Users generally "complete" a form by modifying its controls (entering text, selecting menu items, etc.), before submitting the form to an agent for processing (e.g., to a Web server, to a mail server, etc.)

Here's a simple form that includes labels, radio buttons, and push buttons (reset the form or submit it):

```
<FORM action="http://somesite.com/prog/adduser" method="post" >
<P>
<LABEL for="firstname">First name: </LABEL>
<INPUT type="text" id="firstname"><BR>
<LABEL for="lastname">Last name: </LABEL>
<INPUT type="text" id="lastname"><BR>
<LABEL for="email">email: </LABEL>
<INPUT type="text" id="email"><BR>
<INPUT type="radio" name="sex" value="Male"> Male<BR>
<INPUT type="radio" name="sex" value="Female"> Female<BR>
<INPUT type="submit" value="Send" > <INPUT type="reset" >
</P>
</FORM>
```

Note. This specification includes more detailed information about forms in the subsections on form display issues [p.347] .

17.2 Controls

Users interact with forms through named controls.

A control's "control name" is given by its name attribute. The scope of the name attribute for a control within a FORM element is the FORM element.

Each control has both an initial value and a current value, both of which are character strings. Please consult the definition of each control for information about initial values and possible constraints on values imposed by the control. In general, a control's "initial value" may be specified with the control element's value attribute. However, the initial value of a TEXTAREA element is given by its contents, and the initial value of an OBJECT element in a form is determined by the object implementation (i.e., it lies outside the scope of this specification).

The control's "current value" is first set to the initial value. Thereafter, the control's current value may be modified through user interaction and scripts. [p.251]

A control's initial value does not change. Thus, when a form is reset, each control's current value is reset to its initial value. If a control does not have an initial value, the effect of a form reset on that control is undefined.

When a form is submitted for processing, some controls have their name paired with their current value and these pairs are submitted [p.245] with the form. Those controls for which name/value pairs are submitted are called successful controls [p.245] .

17.2.1 Control types

HTML defines the following control types:

buttons

Authors may create three types of buttons:

- submit buttons: When activated, a submit button submits a form. [p.245] A form may contain more than one submit button.
- reset buttons: When activated, a reset button resets all controls to their initial values. [p.220]
- push buttons: Push buttons have no default behavior. Each push button may have client-side scripts [p.251] associated with the element's event [p.254] attributes. When an event occurs (e.g., the user presses the button, releases it, etc.), the associated script is triggered.

Authors should specify the scripting language of a push button script through a default script declaration [p.253] (with the `META` element).

Authors create buttons with the `BUTTON` element or the `INPUT` element. Please consult the definitions of these elements for details about specifying different button types.

Note. *Authors should note that the `BUTTON` element offers richer rendering capabilities than the `INPUT` element.*

checkboxes

Checkboxes (and radio buttons) are on/off switches that may be toggled by the user. A switch is "on" when the control element's checked attribute is set. When a form is submitted, only "on" checkboxes controls can become successful [p.245].

Several checkboxes in a form may share the same control name. [p.220] Thus, for example, checkboxes allow users to select several values for the same property. The `INPUT` element is used to create a checkbox control.

radio buttons

Radio buttons are like checkboxes except that when several share the same control name [p.220], they are mutually exclusive: when one is switched "on", all others with the same name are switched "off". The `INPUT` element is used to create a radio button control. If no radio button in a set sharing the same control name is initially "on", user agent behavior for choosing which control is initially "on" is undefined. **Note.** Since existing implementations handle this case differently, the current specification differs from RFC 1866 ([RFC1866] [p.356] section 8.1.2.4), which states:

At all times, exactly one of the radio buttons in a set is checked. If none of the `<INPUT>` elements of a set of radio buttons specifies 'CHECKED', then the user agent must check the first radio button of the set initially.

Since user agent behavior differs, authors should ensure that in each set of radio buttons that one is initially "on".

menus

Menus offer users options from which to choose. The `SELECT` element creates a menu, in combination with the `OPTGROUP` and `OPTION` elements.

text input

Authors may create two types of controls that allow users to input text. The `INPUT` element creates a single-line input control and the `TEXTAREA` element creates a multi-line input control. In both cases, the input text becomes the control's current value [p.220].

file select

This control type allows the user to select files so that their contents may be submitted with a form. The `INPUT` element is used to create a file select control.

hidden controls

Authors may create controls that are not rendered but whose values are submitted with a form. Authors generally use this control type to store information between client/server exchanges that would otherwise be lost due to the stateless nature of HTTP (see [RFC2616] [p.354]). The `INPUT` element is used to create a hidden control.

object controls

Authors may insert generic objects in forms such that associated values are submitted along with other controls. Authors create object controls with the `OBJECT` element.

The elements used to create controls generally appear inside a `FORM` element, but may also appear outside of a `FORM` element declaration when they are used to build user interfaces. This is discussed in the section on intrinsic events. [p.254] Note that controls outside a form cannot be successful controls [p.245].

17.3 The `FORM` element

```
<ELEMENT FORM - (%block;|SCRIPT)+ -(FORM) -- interactive form -->
<!ATTLIST FORM
  %attrs;
  action           #REQUIRED -- server-side form handler --
  method          (GET|POST)  GET -- HTTP method used to submit the form--
  enctype         %ContentType; "application/x-www-form-urlencoded"
  accept          %IMPLIED -- list of MIME types for file upload --
  name            CDATA       #IMPLIED -- name of form for scripting --
  onsubmit        %Script;    #IMPLIED -- the form was submitted --
  onreset         %Script;    #IMPLIED -- the form was reset --
  accept-charset %Charset;    #IMPLIED -- list of supported charsets --
  >
```

Start tag: required, End tag: required

Attribute definitions

`action = uri` [p.51] [CT] [p.49]

This attribute specifies a form processing agent. User agent behavior for a value other than an HTTP URI is undefined.

`method = get` | `post` [CI] [p.49]

This attribute specifies which HTTP method will be used to submit the form data set [p.246]. Possible (case-insensitive) values are "get" (the default) and "post". See the section on form submission [p.245] for usage information.

`enctype = content-type` [p.53] [CI] [p.49]

This attribute specifies the content type [p.247] used to submit the form to the server (when the value of `method` is "post"). The default value for this attribute is "application/x-www-form-urlencoded". The value "multipart/form-data" should be used in combination with the `INPUT` element, `type="file"`.

`accept-charset = charset-list` [p.53] [CI] [p.49]

This attribute specifies the list of character encodings [p.41] for input data that is accepted by the server processing this form. The value is a space- and/or comma-delimited list of charset [p.53] values. The client must interpret this list as an exclusive-or list, i.e., the server is able to accept any single character encoding per entity received.

The default value for this attribute is the reserved string "UNKNOWN". User agents may interpret this value as the character encoding that was used to transmit the document containing this `FORM` element.

`accept = content-type-list` [p.53] [CI] [p.49]

This attribute specifies a comma-separated list of content types that a server processing this form will handle correctly. User agents may use this information to filter out non-conforming files when prompting a user to select files to be sent to the server (cf. the `INPUT` element when `type="file"`).

`name = cdata` [p.50] [CI] [p.49]

This attribute names the element so that it may be referred to from style sheets or scripts. **Note.** This attribute has been included for backwards compatibility. Applications should use the `id` attribute to identify elements.

Attributes defined elsewhere

- `id`, `class` (document-wide identifiers [p.71])
- `lang` (language information [p.79]), `dir` (text direction [p.82])
- `style` (inline style information [p.186])
- `title` (element title [p.63])
- `target` (target frame information [p.212])
- `onsubmit`, `onreset`, `onclick`, `ondblclick`, `onmousedown`, `onmouseover`, `onmousemove`, `onmouseout`, `onkeypress`, `onkeydown`, `onkeyup` (intrinsic events [p.254])

The `FORM` element acts as a container for controls [p.220]. It specifies:

- The layout of the form (given by the contents of the element).
- The program that will handle the completed and submitted form (the `action` attribute). The receiving program must be able to parse name/value pairs in order to make use of them.
- The method by which user data will be sent to the server (the `method` attribute).
- A character encoding that must be accepted by the server in order to handle this form (the `accept-charset` attribute). User agents may advise the user of the value of the `accept-charset` attribute and/or restrict the user's ability to enter unrecognized characters.

A form can contain text and markup (paragraphs, lists, etc.) in addition to form controls. [p.220]

The following example shows a form that is to be processed by the "adduser" program when submitted. The form will be sent to the program using the HTTP "post" method.

```
<FORM action="http://someite.com/prog/adduser" method="post">
...Form contents...
</FORM>
```

Please consult the section on form submission [p.245] for information about how user agents must prepare form data for servers and how user agents should handle expected responses.

Note. Further discussion on the behavior of servers that receive form data is beyond the scope of this specification.

17.4 The INPUT element

```
<ENTITY % InputType
" (TEXT | PASSWORD | CHECKBOX |
  RADIO | SUBMIT | RESET |
  FILE | HIDDEN | IMAGE | BUTTON) "
>
```

```
<!-- attribute name required for all but submit and reset -->
<ELEMENT INPUT - O EMPTY
<ATTLIST INPUT
```

```
  %attrs;
  type %InputType;
  name CDATA #IMPLIED
  value CDATA #IMPLIED
  checked (checked) #IMPLIED
  disabled (disabled) #IMPLIED
  readonly (readonly) #IMPLIED
  size CDATA #IMPLIED
  maxLength NUMBER #IMPLIED
  src %URI; #IMPLIED
  alt CDATA #IMPLIED
  usemap (ismap) #IMPLIED
  tabIndex NUMBER #IMPLIED
  accessKey %Character; #IMPLIED
  onfocus %Script; #IMPLIED
  onblur %Script; #IMPLIED

  -- %coreattrs, %i18n, %events --
  -- what kind of widget is needed --
  -- submit as part of form --
  -- Specify for radio buttons and checkboxes --
  -- for radio buttons and check boxes --
  -- unavailable in this context --
  -- for text and passwd --
  -- specific to each type of field --
  -- max chars for text fields --
  -- for fields with images --
  -- short description --
  -- use client-side image map --
  -- use server-side image map --
  -- position in tabbing order --
  -- accessibility key character --
  -- the element got the focus --
  -- the element lost the focus --
```



```

onselect  $Script;      #IMPLIED -- some text was selected --
onchange  $Script;      #IMPLIED -- the element value was changed --
accept    $ContentTypes; #IMPLIED -- list of MIME types for file upload --
>

```

Start tag: **required**, End tag: **forbidden**

Attribute definitions

```

type =
text|password|checkbox|radio|submit|reset|file|hidden|image|button
[CI] [p.49]

```

This attribute specifies the type of control [p.226] to create. The default value for this attribute is "text".

```
name = cdata [p.50] [CI] [p.49]
```

This attribute assigns the control name [p.220].

```
value = cdata [p.50] [CA] [p.49]
```

This attribute specifies the initial value [p.220] of the control. It is optional except when the `type` attribute has the value "radio" or "checkbox".

```
size = cdata [p.50] [CN] [p.49]
```

This attribute tells the user agent the initial width of the control. The width is given in pixels [p.52] except when `type` attribute has the value "text" or "password". In that case, its value refers to the (integer) number of characters. `maxlength = number`[p.50] [CN] [p.49]

When the `type` attribute has the value "text" or "password", this attribute specifies the maximum number of characters the user may enter. This number may exceed the specified `size`, in which case the user agent should offer a scrolling mechanism. The default value for this attribute is an unlimited number. `checked` [CI] [p.49]

When the `type` attribute has the value "radio" or "checkbox", this boolean attribute specifies that the button is on. User agents must ignore this attribute for other control types.

```
src = uri [p.51] [CT] [p.49]
```

When the `type` attribute has the value "image", this attribute specifies the location of the image to be used to decorate the graphical submit button.

Attributes defined elsewhere

- `id`, `class` (document-wide identifiers [p.71])
- `lang` (language information [p.79]), `dir` (text direction [p.82])
- `title` (element title [p.63])
- `style` (inline style information [p.186])
- `alt` (alternate text [p.181])
- `align` (alignment [p.195])
- `accept` (legal content types for a server [p.222])
- `readonly` (read-only input controls [p.244])
- `disabled` (disabled input controls [p.244])
- `tabindex` (tabbing navigation [p.241])

- `accesskey` (access keys [p.242])
- `usemap` (client-side image maps [p.174])
- `ismap` (server-side image maps [p.179])
- `onfocus`, `onblur`, `onselect`, `onchange`, `onclick`, `ondblclick`, `onmousedown`, `onmouseup`, `onmouseover`, `onmousemove`, `onmouseout`, `onkeypress`, `onkeydown`, `onkeyup` (intrinsic events [p.254])

17.4.1 Control types created with INPUT

The control type [p.220] defined by the `INPUT` element depends on the value of the `type` attribute:

text

Creates a single-line text input [p.222] control.

password

Like "text", but the input text is rendered in such a way as to hide the characters (e.g., a series of asterisks). This control type is often used for sensitive input such as passwords. Note that the current value [p.220] is the text entered by the user, not the text rendered by the user agent.

Note. *Application designers should note that this mechanism affords only light security protection. Although the password is masked by user agents from casual observers, it is transmitted to the server in clear text, and may be read by anyone with low-level access to the network.*

checkbox

Creates a checkbox. [p.221]

radio

Creates a radio button. [p.221]

submit

Creates a submit button. [p.221]

image

Creates a graphical submit button. [p.221] The value of the `src` attribute specifies the URI of the image that will decorate the button. For accessibility reasons, authors should provide alternate text [p.181] for the image via the `alt` attribute.

When a pointing device is used to click on the image, the form is submitted and the click coordinates passed to the server. The `x` value is measured in pixels [p.52] from the left of the image, and the `y` value in pixels [p.52] from the top of the image. The submitted data includes `name.x=x-value` and `name.y=y-value` where "name" is the value of the `name` attribute, and `x-value` and `y-value` are the `x` and `y` coordinate values, respectively.

If the server takes different actions depending on the location clicked, users of non-graphical browsers will be disadvantaged. For this reason, authors should consider alternate approaches:

- Use multiple submit buttons (each with its own image) in place of a single graphical submit button. Authors may use style sheets to control the positioning of these buttons.
- Use a client-side image map [p.173] together with scripting.

reset

Creates a reset button. [p.224]

button

Creates a push button. [p.224] User agents should use the value of the value attribute as the button's label.

hidden

Creates a hidden control. [p.222]

file

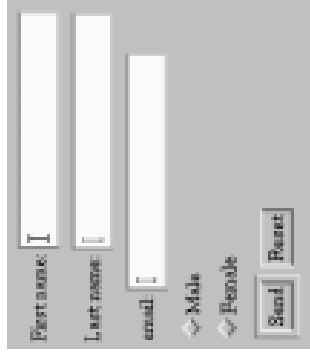
Creates a file select [p.222] control. User agents may use the value of the value attribute as the initial file name.

17.4.2 Examples of forms containing INPUT controls

The following sample HTML fragment defines a simple form that allows the user to enter a first name, last name, email address, and gender. When the submit button is activated, the form will be sent to the program specified by the action attribute.

```
<FORM action="http://somesite.com/prog/adduser" method="post" >
  <P>
    First name: <INPUT type="text" name="firstname"><BR>
    Last name: <INPUT type="text" name="lastname"><BR>
    email: <INPUT type="text" name="email"><BR>
    <INPUT type="radio" name="sex" value="Male" > Male<BR>
    <INPUT type="radio" name="sex" value="Female" > Female<BR>
    <INPUT type="submit" value="Send" /> <INPUT type="reset" />
  </P>
</FORM>
```

This form might be rendered as follows:



In the section on the LABEL element, we discuss marking up labels such as "First name".

In this next example, the JavaScript function name `verify` is triggered when the "onclick" event occurs:

```
<HEAD>
<META http-equiv="Content-Script-Type" content="text/javascript">
</HEAD>
<BODY>
<FORM action="..." method="post">
  <P>
    <INPUT type="button" value="Click Me" onclick="verify()" >
  </FORM>
</BODY>
```

Please consult the section on intrinsic events [p.254] for more information about scripting and events.

The following example shows how the contents of a user-specified file may be submitted with a form. The user is prompted for his or her name and a list of file names whose contents should be submitted with the form. By specifying the enctype value of "multipart/form-data", each file's contents will be packaged for submission in a separate section of a multipart document.

```
<FORM action="http://server.dom/cgi/handle"
  enctype="multipart/form-data"
  method="post" >
  <P>
    What is your name? <INPUT type="text" name="name_of_sender">
    What files are you sending? <INPUT type="file" name="name_of_files">
  </P>
</FORM>
```

17.5 The BUTTON element

```
<!ELEMENT BUTTON - -
  (%flow;)* -(A|%formctrl;|FORM|FIELDSET)
  -- push button -->
<!ATTLIST BUTTON
  %attrs;
  name CDATA #IMPLIED
  value CDATA #IMPLIED
  type (button|submit|reset) submit -- for use as form button --
  disabled (disabled) #IMPLIED -- unavailable in this context --
  tabindex NUMBER #IMPLIED -- position in tabbing order --
  accesskey %character; #IMPLIED -- accessibility key character --
  onfocus %script; #IMPLIED -- the element got the focus --
  onblur %script; #IMPLIED -- the element lost the focus --
  >
```

Start tag: required, End tag: required

Attribute definitions

`name = cdata` [p.50] [C] [p.49]

This attribute assigns the control name. [p.220]

`value = cdata` [p.50] [CS] [p.49]

This attribute assigns the initial value [p.220] to the button.

`type = submit|button|reset` [C] [p.49]

This attribute declares the type of the button. Possible values:

- `submit`: Creates a submit button. [p.221] This is the default value.
- `reset`: Creates a reset button. [p.221]
- `button`: Creates a push button. [p.221]

Attributes defined elsewhere

- `id`, `class` (document-wide identifiers [p.71])
- `lang` (language information [p.79]), `dir` (text direction [p.82])
- `title` (element title [p.63])
- `style` (inline style information [p.186])
- `disabled` (disabled input controls [p.244])
- `accesskey` (access keys [p.242])
- `tabindex` (tabbing navigation [p.241])
- `onmouseover`, `onmousemove`, `onmousedown`, `onmouseup`, `onmouseout`, `onmouseover`, `onmousemove`, `onmousedown`, `onmouseup`, `onmouseout`, `onkeypress`, `onkeydown`, `onkeyup` (intrinsic events [p.254])

Buttons created with the `BUTTON` element function just like buttons created with the `INPUT` element, but they offer richer rendering possibilities: the `BUTTON` element may have content. For example, a `BUTTON` element that contains an image functions like and may resemble an `INPUT` element whose `type` is set to "image", but the `BUTTON` element type allows content.

Visual user agents may render `BUTTON` buttons with relief and an up/down motion when clicked, while they may render `INPUT` buttons as "flat" images.

The following example expands a previous example, but creates submit [p.221] and reset [p.221] buttons with `BUTTON` instead of `INPUT`. The buttons contain images by way of the `IMG` element.

```
<FORM action="http://somesite.com/prog/adduser" method="post">
<P>
First name: <INPUT type="text" name="firstname"><BR>
Last name: <INPUT type="text" name="lastname"><BR>
email: <INPUT type="text" name="email"><BR>
<INPUT type="radio" name="sex" value="Male"> Male<BR>
<INPUT type="radio" name="sex" value="Female"> Female<BR>
<BUTTON name="submit" value="submit" type="submit">
Send<IMG src="/icons/wow.gif" alt="wow"></BUTTON>
```

```
<BUTTON name="reset" type="reset">
Reset<IMG src="/icons/oops.gif" alt="oops"></BUTTON>
</P>
</FORM>
```

Recall that authors must provide alternate text [p.181] for an `IMG` element. It is illegal to associate an image map with an `IMG` that appears as the contents of a `BUTTON` element.

ILLEGAL EXAMPLE:

The following is not legal HTML.

```
<BUTTON>
<IMG src="foo.gif" usemap="...">
</BUTTON>
```

17.6 The `SELECT`, `OPTGROUP`, and `OPTION` elements

```
<ELEMENT SELECT - - (OPTGROUP|OPTION)+ -- option selector -->
<!ATTLIST SELECT
%attrs;
name #IMPLIED -- field name --
size #IMPLIED -- rows visible --
multiple (multiple) #IMPLIED -- default is single selection --
disabled (disabled) #IMPLIED -- unavailable in this context --
tabindex #IMPLIED -- position in tabbing order --
onfocus #IMPLIED -- the element got the focus --
onblur #SCRIPT: #IMPLIED -- the element lost the focus --
onchange #SCRIPT: #IMPLIED -- the element value was changed --
>
```

Start tag: **required**, End tag: **required**

SELECT Attribute definitions

`name = cdata` [p.50] [C] [p.49]

This attribute assigns the control name. [p.220]

`size = number` [p.50] [CN] [p.49]

If a `SELECT` element is presented as a scrolled list box, this attribute specifies the number of rows in the list that should be visible at the same time. Visual user agents are not required to present a `SELECT` element as a list box; they may use any other mechanism, such as a drop-down menu.

`multiple` [C] [p.49]

If set, this boolean attribute allows multiple selections. If not set, the `SELECT` element only permits single selections.

Attributes defined elsewhere

- `id`, `class` (document-wide identifiers [p.71])
- `lang` (language information [p.79]), `dir` (text direction [p.82])
- `title` (element title [p.63])
- `style` (inline style information [p.186])

- `disabled` (disabled input controls [p.244])
- `tabindex` (tabbing navigation [p.241])
- `onclick`, `ondblclick`, `onmousedown`, `onmouseup`, `onmouseover`, `onmousemove`, `onmouseout`, `onkeypress`, `onkeydown`, `onkeyup` (intrinsic events [p.254])

The `SELECT` element creates a menu [p.222] . Each choice offered by the menu is represented by an `OPTION` element. A `SELECT` element must contain at least one `OPTION` element.

The `OPTGROUP` element allows authors to group choices logically. This is particularly helpful when the user must choose from a long list of options; groups of related choices are easier to grasp and remember than a single long list of options. In HTML 4, all `OPTGROUP` elements must be specified directly within a `SELECT` element (i.e., groups may not be nested).

17.6.1 Pre-selected options

Zero or more choices may be pre-selected for the user. User agents should determine which choices are pre-selected as follows:

- If no `OPTION` element has the `selected` attribute set, user agent behavior for choosing which option is initially selected is undefined. **Note.** Since existing implementations handle this case differently, the current specification differs from RFC 1866 ([RFC1866] [p.356] section 8.1.3), which states:

The initial state has the first option selected, unless a `SELECTED` attribute is present on any of the `<OPTION>` elements.

Since user agent behavior differs, authors should ensure that each menu includes a default pre-selected `OPTION`.

- If one `OPTION` element has the `selected` attribute set, it should be pre-selected.
- If the `SELECT` element has the `multiple` attribute set and more than one `OPTION` element has the `selected` attribute set, they should all be pre-selected.
- It is considered an error if more than one `OPTION` element has the `selected` attribute set and the `SELECT` element does not have the `multiple` attribute set. User agents may vary in how they handle this error, but should not pre-select more than one choice.

```
<!ELEMENT OPTGROUP - - ( OPTION)+ - - option group -->
<!ATTLIST OPTGROUP
  %attrs;
  disabled (disabled) #IMPLIED -- unavailable in this context --
  label    %Text;      #REQUIRED -- for use in hierarchical menus --
>
```

Start tag: required, End tag: required

OPTGROUP Attribute definitions

`label = text` [p.50] [CS] [p.49]

This attribute specifies the label for the option group.

Attributes defined elsewhere

- `id`, `class` (document-wide identifiers [p.71])
- `lang` (language information [p.79]), `dir` (text direction [p.82])
- `title` (element title [p.63])
- `style` (inline style information [p.186])
- `disabled` (disabled input controls [p.244])
- `onclick`, `ondblclick`, `onmousedown`, `onmouseup`, `onmouseover`, `onmousemove`, `onmouseout`, `onkeypress`, `onkeydown`, `onkeyup` (intrinsic events [p.254])

Note. Implementors are advised that future versions of HTML may extend the grouping mechanism to allow for nested groups (i.e., *OPTGROUP* elements may nest). This will allow authors to represent a richer hierarchy of choices.

```
<!ELEMENT OPTION - O (#PCDATA) -- selectable choice -->
<!ATTLIST OPTION
  %attrs;
  selected (selected) #IMPLIED -- %coreattrs, %i18n, %events --
  disabled (disabled) #IMPLIED -- unavailable in this context --
  label    %Text;      #IMPLIED -- for use in hierarchical menus --
  value    CDATA       #IMPLIED -- defaults to element content --
>
```

Start tag: required, End tag: optional

OPTION Attribute definitions

`selected` [C] [p.49]

When set, this boolean attribute specifies that this option is pre-selected.

`value = cdata` [p.50] [CS] [p.49]

This attribute specifies the initial value [p.220] of the control. If this attribute is not set, the initial value [p.220] is set to the contents of the `OPTION` element.

`label = text` [p.50] [CS] [p.49]

This attribute allows authors to specify a shorter label for an option than the content of the `OPTION` element. When specified, user agents should use the value of this attribute rather than the content of the `OPTION` element as the option label.

Attributes defined elsewhere

- `id`, `class` (document-wide identifiers [p.71])
- `lang` (language information [p.79]), `dir` (text direction [p.82])

- **title** (element title [p.63])
- **style** (inline style information [p.186])
- **disabled** (disabled input controls [p.244])
- **onclick, ondblclick, onmousedown, onmouseover, onmousemove, onmouseout, onkeypress, onkeydown, onkeyup** (intrinsic events [p.254])

When rendering a menu choice, user agents should use the value of the `label` attribute of the `OPTION` element as the choice. If this attribute is not specified, user agents should use the contents of the `OPTION` element.

The `label` attribute of the `OPTION` element specifies the label for a group of choices.

In this example, we create a menu that allows the user to select which of seven software components to install. The first and second components are pre-selected but may be deselected by the user. The remaining components are not pre-selected. The `size` attribute states that the menu should only have 4 rows even though the user may select from among 7 options. The other options should be made available through a scrolling mechanism.

The `SELECT` is followed by submit and reset buttons.

```
<FORM action="http://somesite.com/program/component-select" method="post">
<P>
<SELECT multiple size="4" name="component-select">
<OPTION selected value="Component_1_a">Component_1</OPTION>
<OPTION>Component_2</OPTION>
<OPTION>Component_3</OPTION>
<OPTION>Component_4</OPTION>
<OPTION>Component_5</OPTION>
<OPTION>Component_6</OPTION>
<OPTION>Component_7</OPTION>
</SELECT>
<INPUT type="submit" value="Send"><INPUT type="reset" >
</P>
</FORM>
```

Only selected options will be successful [p.245] (using the control name [p.220] "component-select"). When no options are selected, the control is not successful and neither the name nor any values are submitted to the server when the form is submitted. Note that where the value attribute is set, it determines the control's initial value [p.220], otherwise it's the element's contents.

In this example we use the `OPTION` element to group choices. The following markup:

```
<FORM action="http://somesite.com/program/someprog" method="post">
<P>
<SELECT name="ComOS">
<OPTION selected label="none" value="none">None</OPTION>
<OPTION label="PortMaster 3">
<OPTION label="3.7.1" value="pm3_3.7.1">PortMaster 3 with ComOS 3.7.1</OPTION>
<OPTION label="3.7" value="pm3_3.7">PortMaster 3 with ComOS 3.7</OPTION>
```

```
<OPTION label="3.5" value="pm3_3.5">PortMaster 3 with ComOS 3.5</OPTION>
</OPTIONGROUP>
<OPTION label="PortMaster 2">
<OPTION label="3.7" value="pm2_3.7">PortMaster 2 with ComOS 3.7</OPTION>
<OPTION label="3.5" value="pm2_3.5">PortMaster 2 with ComOS 3.5</OPTION>
</OPTIONGROUP>
<OPTION label="IRX">
<OPTION label="3.7R" value="IRX_3.7R">IRX with ComOS 3.7R</OPTION>
<OPTION label="3.5R" value="IRX_3.5R">IRX with ComOS 3.5R</OPTION>
</OPTIONGROUP>
</SELECT>
</FORM>
```

represents the following grouping:

```
None
PortMaster 3
 3.7.1
 3.7
 3.5
PortMaster 2
 3.7
 3.5
IRX
 3.7R
 3.5R
```

Visual user agents may allow users to select from option groups through a hierarchical menu or some other mechanism that reflects the structure of choices. A graphical user agent might render this as:



This image shows a `SELECT` element rendered as cascading menus. The top label of the menu displays the currently selected value (PortMaster 3, 3.7.1). The user has unfurled two cascading menus, but has not yet selected the new value (PortMaster 2, 3.7). Note that each cascading menu displays the label of an `OPTION` or `OPTION` element.

17.7 The `TEXTAREA` element

```
<ELEMENT TEXTAREA - - (#PCDATA) -- multi-line text field -->
<!ATTLIST TEXTAREA
%attrs;
name CDATA #IMPLIED
rows NUMBER #REQUIRED
cols NUMBER #REQUIRED
disabled (disabled) #IMPLIED -- unavailable in this context --
readonly (readonly) #IMPLIED
tabindex NUMBER #IMPLIED -- position in tabbing order --
```

```

accesskey %Character; #IMPLIED -- accessibility key character --
onfocus %Script; #IMPLIED -- the element got the focus --
onblur %Script; #IMPLIED -- the element lost the focus --
onselect %Script; #IMPLIED -- some text was selected --
onchange %Script; #IMPLIED -- the element value was changed --
>

```

Start tag: **required**, End tag: **required**

Attribute definitions

`name = cdata` [p.50] [C:] [p.49]

This attribute assigns the control name. [p.220]

`rows = number` [p.50] [CN] [p.49]

This attribute specifies the number of visible text lines. Users should be able to enter more lines than this, so user agents should provide some means to scroll through the contents of the control when the contents extend beyond the visible area.

`cols = number` [p.50] [CN] [p.49]

This attribute specifies the visible width in average character widths. Users should be able to enter longer lines than this, so user agents should provide some means to scroll through the contents of the control when the contents extend beyond the visible area. User agents may wrap visible text lines to keep long lines visible without the need for scrolling.

Attributes defined elsewhere

- `id, class` (document-wide identifiers [p.71])
- `lang` (language information [p.79]), `dir` (text direction [p.82])
- `title` (element title [p.63])
- `style` (inline style information [p.186])
- `readonly` (read-only input controls [p.244])
- `disabled` (disabled input controls [p.244])
- `tabindex` (tabbing navigation [p.241])
- `onfocus, onblur, onselect, onchange, onclick, ondblclick, onmousedown, onmouseover, onmouseout, onmousemove, onmouseup, onkeydown, onkeyup` (intrinsic events [p.254])

The `TEXTAREA` element creates a multi-line text input [p.222] control. User agents should use the contents of this element as the initial value [p.220] of the control and should render this text initially.

This example creates a `TEXTAREA` control that is 20 rows by 80 columns and contains two lines of text initially. The `TEXTAREA` is followed by submit and reset buttons.

```

<FORM action="http://some.site.com/prog/text-read" method="post">
<P>
<TEXTAREA name="thetext" rows="20" cols="80">
First line of initial text.
Second line of initial text.
</TEXTAREA>
<INPUT type="submit" value="Send"><INPUT type="reset">
</P>
</FORM>

```

Setting the `readonly` attribute allows authors to display unmodifiable text in a `TEXTAREA`. This differs from using standard marked-up text in a document because the value of `TEXTAREA` is submitted with the form.

17.8 The `ISINDEX` element

`ISINDEX` is deprecated [p.38]. This element creates a single-line text input [p.222] control. Authors should use the `INPUT` element to create text input [p.222] controls.

See the Transitional DTD [p.294] for the formal definition.

Attribute definitions

`prompt = text` [p.50] [CS] [p.49]

Deprecated. [p.38] This attribute specifies a prompt string for the input field.

Attributes defined elsewhere

- `id, class` (document-wide identifiers [p.71])
- `lang` (language information [p.79]), `dir` (text direction [p.82])
- `title` (element title [p.63])
- `style` (inline style information [p.186])

The `ISINDEX` element creates a single-line text input [p.222] control that allows any number of characters. User agents may use the value of the `prompt` attribute as a title for the prompt.

DEPRECATED EXAMPLE:

The following `ISINDEX` declaration:

```
<ISINDEX prompt="Enter your search phrase: ">
```

could be rewritten with `INPUT` as follows:

```
<FORM action="..." method="post">
<P>Enter your search phrase: <INPUT type="text"></P>
</FORM>
```

Semantics of `ISINDEX`. Currently, the semantics for `ISINDEX` are only well-defined when the base URI for the enclosing document is an HTTP URI. In practice, the input string is restricted to Latin-1 as there is no mechanism for the URI to specify a different character set.

17.9 Labels

Some form controls automatically have labels associated with them (press buttons) while most do not (text fields, checkboxes and radio buttons, and menus).

For those controls that have implicit labels, user agents should use the value of the value attribute as the label string.

The LABEL element is used to specify labels for controls that do not have implicit labels,

17.9.1 The LABEL element

```
<!ELEMENT LABEL - - (%inline;)* -(LABEL) -- form field label text -->
<!ATTLIST LABEL
  %attrs;
  for IDREF #IMPLIED -- matches field ID value --
  accesskey %Character; #IMPLIED -- accessibility key character --
  onfocus %Script; #IMPLIED -- the element got the focus --
  onblur %Script; #IMPLIED -- the element lost the focus --
  >
```

Start tag: required, End tag: required

Attribute definitions

`for` = *idref* [p.50] [CS] [p.49]

This attribute explicitly associates the label being defined with another control. When present, the value of this attribute must be the same as the value of the `id` attribute of some other control in the same document. When absent, the label being defined is associated with the element's contents.

Attributes defined elsewhere

- `id`, class (document-wide identifiers [p.71])
- lang (language information [p.79]), dir (text direction [p.82])
- title (element title [p.63])
- style (inline style information [p.186])
- accesskey (access keys [p.242])
- onfocus, onblur, onclick, ondblclick, onmousedown, onmouseup, onmouseover, onmousemove, onmouseout, onkeypress, onkeydown, onkeyup (intrinsic events [p.254])

The LABEL element may be used to attach information to controls. Each LABEL element is associated with exactly one form control.

The `for` attribute associates a label with another control explicitly: the value of the `for` attribute must be the same as the value of the `id` attribute of the associated control element. More than one LABEL may be associated with the same control by creating multiple references via the `for` attribute.

This example creates a table that is used to align two text input [p.222] controls and their associated labels. Each label is associated explicitly with one text input [p.222]:

```
<FORM action="..." method="post">
<TABLE>
<TR>
<TD><LABEL for="fname">First Name</LABEL>
<TD><INPUT type="text" name="fname" id="fname">
<TR>
<TD><LABEL for="lname">Last Name</LABEL>
<TD><INPUT type="text" name="lname" id="lname">
</TABLE>
</FORM>
```

This example extends a previous example form to include LABEL elements.

```
<FORM action="http://some.site.com/prog/adduser" method="post">
<P>
<LABEL for="firstname">First name: </LABEL>
<INPUT type="text" id="firstname"><BR>
<LABEL for="lastname">Last name: </LABEL>
<INPUT type="text" id="lastname"><BR>
<LABEL for="email">email: </LABEL>
<INPUT type="text" id="email"><BR>
<INPUT type="radio" name="sex" value="Male"> Male<BR>
<INPUT type="radio" name="sex" value="Female"> Female<BR>
<INPUT type="submit" value="Send"> <INPUT type="reset">
</P>
</FORM>
```

To associate a label with another control implicitly, the control element must be within the contents of the LABEL element. In this case, the LABEL may only contain one control element. The label itself may be positioned before or after the associated control.

In this example, we implicitly associate two labels with two text input [p.222] controls:

```
<FORM action="..." method="post">
<P>
<LABEL>
  First Name
  <INPUT type="text" name="firstname">
</LABEL>
<LABEL>
  Last Name
  <INPUT type="text" name="lastname">
</LABEL>
</P>
</FORM>
```

Note that this technique cannot be used when a table is being used for layout, with the label in one cell and its associated control in another cell.

When a LABEL element receives focus [p.241], it passes the focus on to its associated control. See the section below on access keys [p.242] for examples. Labels may be rendered by user agents in a number of ways (e.g., visually, read by speech synthesizers, etc.)

17.10 Adding structure to forms: the FIELDSET and LEGEND elements

```
<!--
#PCDATA is to solve the mixed content problem,
per specification only whitespace is allowed there!
-->
<!ELEMENT FIELDSET - - (#PCDATA,LEGEND,({flow?}) * ) -- form control group -->
<!ATTLIST FIELDSET
  %attrs;
  >
<!ELEMENT LEGEND - - ({inline;} *
  -- fieldset legend -->
<!--
<!ATTLIST LEGEND
  %attrs;
  %coreattrs, %i18n, %events --
  %accesskey %Character; #IMPLIED -- accessibility key character --
  >
```

Start tag: required. End tag: required

LEGEND Attribute definitions

`align = top|bottom|left|right [C] [p.49]`

Deprecated. [p.38] This attribute specifies the position of the legend with respect to the fieldset. Possible values:

- **top:** The legend is at the top of the fieldset. This is the default value.
- **bottom:** The legend is at the bottom of the fieldset.
- **left:** The legend is at the left side of the fieldset.
- **right:** The legend is at the right side of the fieldset.

Attributes defined elsewhere

- **id, class** (document-wide identifiers [p.71])
- **lang** (language information [p.79]), **dir** (text direction [p.82])
- **title** (element title [p.63])
- **style** (inline style information [p.186])
- **accesskey** (access keys [p.242])
- **onclick, ondblclick, onmousedown, onmouseup, onmouseover, onmousemove, onmouseout, onkeypress, onkeydown, onkeyup** (intrinsic events [p.254])

The FIELDSET element allows authors to group thematically related controls and labels. Grouping controls makes it easier for users to understand their purpose while simultaneously facilitating tabbing navigation for visual user agents and speech navigation for speech-oriented user agents. The proper use of this element makes documents more accessible.

The LEGEND element allows authors to assign a caption to a FIELDSET. The legend improves accessibility when the FIELDSET is rendered non-visually.

In this example, we create a form that one might fill out at the doctor's office. It is divided into three sections: personal information, medical history, and current medication. Each section contains controls for inputting the appropriate information.

```
<FORM action="..." method="post">
<P>
<FIELDSET>
<LEGEND>Personal Information</LEGEND>
Last Name: <INPUT name="personal_lastname" type="text" tabindex="1">
First Name: <INPUT name="personal_firstname" type="text" tabindex="2">
Address: <INPUT name="personal_address" type="text" tabindex="3">
...more personal information...
</FIELDSET>
<FIELDSET>
<LEGEND>Medical History</LEGEND>
<INPUT name="history_illness"
  type="checkbox"
  value="Smallpox" tabindex="20"> Smallpox
  type="checkbox"
  value="Mumps" tabindex="21"> Mumps
<INPUT name="history_illness"
  type="checkbox"
  value="Dizziness" tabindex="22"> Dizziness
<INPUT name="history_illness"
  type="checkbox"
  value="Sneezing" tabindex="23"> Sneezing
...more medical history...
</FIELDSET>
<LEGEND>Current Medication</LEGEND>
Are you currently taking any medication?
<INPUT name="medication_now"
  type="radio"
  value="Yes" tabindex="35">Yes
<INPUT name="medication_now"
  type="radio"
  value="No" tabindex="35">No
If you are currently taking medication, please indicate
it in the space below:
<TEXTAREA name="current_medication"
  rows="20" cols="50"
  tabindex="40">
</TEXTAREA>
</FIELDSET>
</FORM>
```


Note that in this example, we might improve the visual presentation of the form by aligning elements within each `FIELDSET` (with style sheets), adding color and font information (with style sheets), adding scripting (say, to only open the "current medication" text area if the user indicates he or she is currently on medication), etc.

17.11 Giving focus to an element

In an HTML document, an element must receive focus from the user in order to become active and perform its tasks. For example, users must activate a link specified by the `A` element in order to follow the specified link. Similarly, users must give a `TEXTAREA` focus in order to enter text into it.

There are several ways to give focus to an element:

- Designate the element with a pointing device.
- Navigate from one element to the next with the keyboard. The document's author may define a *tabbing order* that specifies the order in which elements will receive focus if the user navigates the document with the keyboard (see tabbing navigation [p.241]). Once selected, an element may be activated by some other key sequence.
- Select an element through an access key [p.242] (sometimes called "keyboard shortcut" or "keyboard accelerator").

17.11.1 Tabbing navigation

Attribute definitions

`tabindex = number` [p.50] [CN] [p.49]

This attribute specifies the position of the current element in the tabbing order for the current document. This value must be a number between 0 and 32767. User agents should ignore leading zeros.

The *tabbing order* defines the order in which elements will receive focus when navigated by the user via the keyboard. The tabbing order may include elements nested within other elements.

Elements that may receive focus should be navigated by user agents according to the following rules:

1. Those elements that support the `tabindex` attribute and assign a positive value to it are navigated first. Navigation proceeds from the element with the lowest `tabindex` value to the element with the highest value. Values need not be sequential nor must they begin with any particular value. Elements that have identical `tabindex` values should be navigated in the order they appear in the character stream.
2. Those elements that do not support the `tabindex` attribute or support it and assign it a value of "0" are navigated next. These elements are navigated in the order they appear in the character stream.

3. Elements that are disabled [p.244] do not participate in the tabbing order.

The following elements support the `tabindex` attribute: `A`, `AREA`, `BUTTON`, `INPUT`, `OBJECT`, `SELECT`, and `TEXTAREA`.

In this example, the tabbing order will be the `BUTTON`, the `INPUT` elements in order (note that "field1" and the button share the same `tabindex`, but "field1" appears later in the character stream), and finally the link created by the `A` element.

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01//EN"
"http://www.w3.org/TR/html4/strict.dtd">
<HTML>
<HEAD>
<TITLE>A document with FORM</TITLE>
</HEAD>
<BODY>
...some text...
<P>Go to the
<A tabindex="10" href="http://www.w3.org/">W3C Web site.</A>
...some more...
<BUTTON type="button" name="get-database"
tabindex="1" onclick="get-database">
Get the current database.
</BUTTON>
...some more...
<FORM action="..." method="post">
<P>
<INPUT tabindex="1" type="text" name="field1">
<INPUT tabindex="2" type="text" name="field2">
<INPUT tabindex="3" type="submit" name="submit">
</P>
</FORM>
</BODY>
</HTML>
```

Tabbing keys. *The actual key sequence that causes tabbing navigation or element activation depends on the configuration of the user agent (e.g., the "tab" key is used for navigation and the "enter" key is used to activate a selected element).*

User agents may also define key sequences to navigate the tabbing order in reverse. When the end (or beginning) of the tabbing order is reached, user agents may circle back to the beginning (or end).

17.11.2 Access keys

Attribute definitions

`accesskey = character` [p.53] [CN] [p.49]

This attribute assigns an access key to an element. An access key is a single character from the document character set. **Note.** Authors should consider the input method of the expected reader when specifying an accesskey.

Pressing an access key assigned to an element gives focus to the element. The action that occurs when an element receives focus depends on the element. For example, when a user activates a link defined by the `A` element, the user agent generally follows the link. When a user activates a radio button, the user agent changes the value of the radio button. When the user activates a text field, it allows input, etc.

The following elements support the `accesskey` attribute: `A`, `AREA`, `BUTTON`, `INPUT`, `LABEL`, and `LEGEND`, and `TEXTAREA`.

This example assigns the access key "U" to a label associated with an `INPUT` control. Typing the access key gives focus to the label which in turn gives it to the associated control. The user may then enter text into the `INPUT` area.

```
<FORM action="..." method="post">
<P>
<LABEL for="fuser" accesskey="U">
  User Name
</LABEL>
<INPUT type="text" name="user" id="fuser">
</P>
</FORM>
```

In this example, we assign an access key to a link defined by the `A` element. Typing this access key takes the user to another document, in this case, a table of contents.

```
<P><A accesskey="C"
  rel="contents"
  href="http://someplace.com/specification/contents.html">
  Table of Contents</A>
```

The invocation of access keys depends on the underlying system. For instance, on machines running MS Windows, one generally has to press the "alt" key in addition to the access key. On Apple systems, one generally has to press the "cmd" key in addition to the access key.

The rendering of access keys depends on the user agent. We recommend that authors include the access key in label text or wherever the access key is to apply. User agents should render the value of an access key in such a way as to emphasize its role and to distinguish it from other characters (e.g., by underlining it).

17.12 Disabled and read-only controls

In contexts where user input is either undesirable or irrelevant, it is important to be able to disable a control or render it read-only. For example, one may want to disable a form's submit button until the user has entered some required data. Similarly, an author may want to include a piece of read-only text that must be submitted as a value along with the form. The following sections describe disabled and read-only controls.

17.12.1 Disabled controls

Attribute definitions

`disabled` [C] [p.49]

When set for a form control, this boolean attribute disables the control for user input.

When set, the `disabled` attribute has the following effects on an element:

- Disabled controls do not receive focus [p.241].
- Disabled controls are skipped in tabbing navigation [p.241].
- Disabled controls cannot be successful [p.245].

The following elements support the `disabled` attribute: `BUTTON`, `INPUT`, `OPTGROUP`, `OPTION`, `SELECT`, and `TEXTAREA`.

This attribute is inherited but local declarations override the inherited value.

How disabled elements are rendered depends on the user agent. For example, some user agents "gray out" disabled menu items, button labels, etc.

In this example, the `INPUT` element is disabled. Therefore, it cannot receive user input nor will its value be submitted with the form.

```
<INPUT disabled name="fred" value="stone">
```

Note. The only way to modify dynamically the value of the `disabled` attribute is through a script. [p.251]

17.12.2 Read-only controls

Attribute definitions

`readonly` [C] [p.49]

When set for a form control, this boolean attribute prohibits changes to the control.

The `readonly` attribute specifies whether the control may be modified by the user.

When set, the `readonly` attribute has the following effects on an element:

- Read-only elements receive focus [p.241] but cannot be modified by the user.
- Read-only elements are included in tabbing navigation [p.241].
- Read-only elements may be successful [p.245].

The following elements support the `readonly` attribute: `INPUT` and `TEXTAREA`.

How read-only elements are rendered depends on the user agent.

Note. *The only way to modify dynamically the value of the `readonly` attribute is through a script.* [p.251]

17.13 Form submission

The following sections explain how user agents submit form data to form processing agents.

17.13.1 Form submission method

The `method` attribute of the `FORM` element specifies the HTTP method used to send the form to the processing agent. This attribute may take two values:

- `get`: With the HTTP "get" method, the form data set [p.246] is appended to the URI specified by the `action` attribute (with a question-mark ("?") as separator) and this new URI is sent to the processing agent.
- `post`: With the HTTP "post" method, the form data set [p.246] is included in the body of the form and sent to the processing agent.

The "get" method should be used when the form is idempotent (i.e., causes no side-effects). Many database searches have no visible side-effects and make ideal applications for the "get" method.

If the service associated with the processing of a form causes side effects (for example, if the form modifies a database or subscription to a service), the "post" method should be used.

Note. *The "get" method restricts form data set [p.246] values to ASCII characters. Only the "post" method (with `enctype="multipart/form-data"`) is specified to cover the entire [ISO10646] [p.353] character set.*

17.13.2 Successful controls

A *successful control* is "valid" for submission. Every successful control has its control name [p.220] paired with its current value [p.220] as part of the submitted form data set [p.246]. A successful control must be defined within a `FORM` element and must have a control name. [p.220]

However:

- Controls that are disabled [p.244] cannot be successful.
- If a form contains more than one submit button [p.221], only the activated submit button is successful.
- All "on" checkboxes [p.221] may be successful.
- For radio buttons [p.221] that share the same value of the `name` attribute, only the "on" radio button may be successful.
- For menus [p.222], the control name [p.220] is provided by a `SELECT` element

and values are provided by `OPTION` elements. Only selected options may be successful. When no options are selected, the control is not successful and neither the name nor any values are submitted to the server when the form is submitted.

- The current value [p.220] of a file select [p.222] is a list of one or more file names. Upon submission of the form, the *contents* of each file are submitted with the rest of the form data. The file contents are packaged according to the form's content type [p.247].
- The current value of an object control is determined by the object's implementation.

If a control doesn't have a current value [p.220] when the form is submitted, user agents are not required to treat it as a successful control.

Furthermore, user agents should not consider the following controls successful:

- Reset buttons. [p.221]
- `OBJECT` elements whose `declare` attribute has been set.

Hidden controls [p.222] and controls that are not rendered because of style sheet [p.183] settings may still be successful. For example:

```
<FORM action="..." method="post">
<P>
<INPUT type="password" style="display:none"
       name="invisible-password"
       value="mypassword">
</FORM>
```

will still cause a value to be paired with the name "invisible-password" and submitted with the form.

17.13.3 Processing form data

When the user submits a form (e.g., by activating a submit button [p.221]), the user agent processes it as follows.

Step one: Identify the successful controls

Step two: Build a form data set

A *form data set* is a sequence of control-name [p.220] /current-value [p.220] pairs constructed from successful controls [p.245]

Step three: Encode the form data set

The form data set is then encoded according to the content type [p.247] specified by the `enctype` attribute of the `FORM` element.

Step four: Submit the encoded form data set

Finally, the encoded data is sent to the processing agent designated by the `action` attribute using the protocol specified by the `method` attribute.

This specification does not specify all valid submission methods or content types [p.247] that may be used with forms. However, HTML 4 user agents must support the established conventions in the following cases:

- If the `method` is "get" and the `action` is an HTTP URI, the user agent takes the value of `action`, appends a '?' to it, then appends the form data set [p.246], encoded using the "application/x-www-form-urlencoded" content type [p.247]. The user agent then traverses the link to this URI. In this scenario, form data are restricted to ASCII codes.
- If the `method` is "post" and the `action` is an HTTP URI, the user agent conducts an HTTP "post" transaction using the value of the `action` attribute and a message created according to the content type [p.247] specified by the `enctype` attribute.

For any other value of `action` or `method`, behavior is unspecified.

User agents should render the response from the HTTP "get" and "post" transactions.

17.13.4 Form content types

The `enctype` attribute of the `FORM` element specifies the content type [p.53] used to encode the form data set [p.246] for submission to the server. User agents must support the content types listed below. Behavior for other content types is unspecified.

Please also consult the section on escaping ampersands in URI attribute values [p.335].

application/x-www-form-urlencoded

This is the default content type. Forms submitted with this content type must be encoded as follows:

1. Control names and values are escaped. Space characters are replaced by '+', and then reserved characters are escaped as described in [RFC1738] [p.354], section 2.2. Non-alphanumeric characters are replaced by '%HH', a percent sign and two hexadecimal digits representing the ASCII code of the character. Line breaks are represented as "CR LF" pairs (i.e., '\%0D%0A').
2. The control names/values are listed in the order they appear in the document. The name is separated from the value by '=' and name/value pairs are separated from each other by '&'.

multipart/form-data

Note. Please consult [RFC2388] [p.356] for additional information about file uploads, including backwards compatibility issues, the relationship between "multipart/form-data" and other content types, performance issues, etc.

Please consult the appendix for information about security issues for forms [p.350]

The content type "application/x-www-form-urlencoded" is inefficient for sending large quantities of binary data or text containing non-ASCII characters. The content type "multipart/form-data" should be used for submitting forms that contain files, non-ASCII data, and binary data.

The content "multipart/form-data" follows the rules of all multipart MIME data streams as outlined in [RFC2045] [p.354]. The definition of "multipart/form-data" is available at the [IANA] [p.353] registry.

A "multipart/form-data" message contains a series of parts, each representing a successful control [p.245]. The parts are sent to the processing agent in the same order the corresponding controls appear in the document stream. Part boundaries should not occur in any of the data; how this is done lies outside the scope of this specification.

As with all multipart MIME types, each part has an optional "Content-Type" header that defaults to "text/plain". User agents should supply the "Content-Type" header, accompanied by a "charset" parameter.

Each part is expected to contain:

1. a "Content-Disposition" header whose value is "form-data".
2. a name attribute specifying the control name [p.220] of the corresponding control. Control names originally encoded in non-ASCII character sets [p.41] may be encoded using the method outlined in [RFC2045] [p.354].

Thus, for example, for a control named "mycontrol", the corresponding part would be specified:

```
Content-Disposition: form-data; name="mycontrol"
```

As with all MIME transmissions, "CR LF" (i.e., '\%0D%0A') is used to separate lines of data.

Each part may be encoded and the "Content-Transfer-Encoding" header supplied if the value of that part does not conform to the default (7BIT) encoding (see [RFC2045] [p.354], section 6)

If the contents of a file are submitted with a form, the file input should be identified by the appropriate content type [p.53] (e.g., "application/octet-stream"). If multiple files are to be returned as the result of a single form entry, they should be returned as "multipart/mixed" embedded within the "multipart/form-data".

The user agent should attempt to supply a file name for each submitted file. The file name may be specified with the "filename" parameter of the 'Content-Disposition: form-data' header, or, in the case of multiple files, in a 'Content-Disposition: file' header of the subpart. If the file name of the client's operating system is not in US-ASCII, the file name might be approximated or encoded using the method of [RFC2045] [p.354]. This is convenient for those cases where, for example, the uploaded files might contain references to each other (e.g., a TeX file and its ".sty" auxiliary style description).

The following example illustrates "multipart/form-data" encoding. Suppose we have the following form:

```
<FORM action="http://server.com/cgi/handle"
  enctype="multipart/form-data"
  method="post">
<P>
  What is your name? <INPUT type="text" name="submit-name"><BR>
  What files are you sending? <INPUT type="file" name="files"><BR>
  <INPUT type="submit" value="Send"> <INPUT type="reset">
</FORM>
```

If the user enters "Larry" in the text input, and selects the text file "file1.txt", the user agent might send back the following data:

```
Content-Type: multipart/form-data; boundary=AaB03x
--AaB03x
Content-Disposition: form-data; name="submit-name"

Larry
--AaB03x
Content-Disposition: form-data; name="files"; filename="file1.txt"
Content-Type: text/plain

... contents of file1.txt ...
--AaB03x--
```

If the user selected a second (image) file "file2.gif", the user agent might construct the parts as follows:

```
Content-Type: multipart/form-data; boundary=AaB03x
--AaB03x
Content-Disposition: form-data; name="submit-name"

Larry
--AaB03x
Content-Disposition: form-data; name="files"
Content-Type: multipart/mixed; boundary=BbC04y

--BbC04y
Content-Disposition: file; filename="file1.txt"
Content-Type: text/plain

... contents of file1.txt ...
```

```
--BbC04y
Content-Disposition: file; filename="file2.gif"
Content-Type: image/gif
Content-Transfer-Encoding: binary

... contents of file2.gif ...
--BbC04y--
--AaB03x--
```

18 Scripts

Contents

1. Introduction to scripts	251
2. Designing documents for user agents that support scripting	252
1. The <code>SCRIPT</code> element	252
2. Specifying the scripting language	253
• The default scripting language	253
• Local declaration of a scripting language	254
• References to HTML elements from a script	254
3. Intrinsic events	254
4. Dynamic modification of documents	258
3. Designing documents for user agents that don't support scripting	258
1. The <code>NOSCRIPT</code> element	258
2. Hiding script data from user agents	259

18.1 Introduction to scripts

A client-side *script* is a program that may accompany an HTML document or be embedded directly in it. The program executes on the client's machine when the document loads, or at some other time such as when a link is activated. HTML's support for scripts is independent of the scripting language.

Scripts offer authors a means to extend HTML documents in highly active and interactive ways. For example:

- Scripts may be evaluated as a document loads to modify the contents of the document dynamically.
- Scripts may accompany a form to process input as it is entered. Designers may dynamically fill out parts of a form based on the values of other fields. They may also ensure that input data conforms to predetermined ranges of values, that fields are mutually consistent, etc.
- Scripts may be triggered by events that affect the document, such as loading, unloading, element focus, mouse movement, etc.
- Scripts may be linked to form controls (e.g., buttons) to produce graphical user interface elements.

There are two types of scripts authors may attach to an HTML document:

- Those that are executed one time when the document is loaded by the user agent. Scripts that appear within a `SCRIPT` element are executed when the document is loaded. For user agents that cannot or will not handle scripts, authors may include alternate content via the `NOSCRIPT` element.
- Those that are executed every time a specific event occurs. These scripts may be assigned to a number of elements via the intrinsic event [p.254] attributes.

Note. This specification includes more detailed information about scripting in sections on script macros [p.348].

18.2 Designing documents for user agents that support scripting

The following sections discuss issues that concern user agents that support scripting.

18.2.1 The `SCRIPT` element

```
<!ELEMENT SCRIPT - - %Script; -- script statements -->
<!ATTLIST SCRIPT
  charset %Charset; #IMPLIED -- char encoding of linked resource --
  type %ContentType; #REQUIRED -- content type of script language --
  src %URI; #IMPLIED -- URI for an external script --
  defer (defer) #IMPLIED -- UA may defer execution of script --
  >
```

Start tag: required, End tag: required

Attribute definitions

`src = uri` [p.51] [CT] [p.49]

This attribute specifies the location of an external script.

`type = content-type` [p.53] [CI] [p.49]

This attribute specifies the scripting language of the element's contents and overrides the default scripting language. The scripting language is specified as a content type (e.g., "text/javascript"). Authors must supply a value for this attribute. There is no default value for this attribute.

`language = cdata` [p.50] [CI] [p.49]

Deprecated. [p.38] This attribute specifies the scripting language of the contents of this element. Its value is an identifier for the language, but since these identifiers are not standard, this attribute has been deprecated [p.38] in favor of `type`.

`defer` [CI] [p.49]

When set, this boolean attribute provides a hint to the user agent that the script is not going to generate any document content (e.g., no "document.write" in javascript) and thus, the user agent can continue parsing and rendering.

Attributes defined elsewhere

- `charset`(character encodings [p.41])

The `SCRIPT` element places a script within a document. This element may appear any number of times in the `HEAD` or `BODY` of an HTML document.

The script may be defined within the contents of the `SCRIPT` element or in an external file. If the `src` attribute is not set, user agents must interpret the contents of the element as the script. If the `src` has a URI value, user agents must ignore the element's contents and retrieve the script via the URI. Note that the `charset` attribute refers to the character encoding [p.41] of the script designated by the `src` attribute; it does not concern the content of the `SCRIPT` element.

Scripts are evaluated by *script engines* that must be known to a user agent.

The syntax of script data [p.57] depends on the scripting language.

18.2.2 Specifying the scripting language

As HTML does not rely on a specific scripting language, document authors must explicitly tell user agents the language of each script. This may be done either through a default declaration or a local declaration.

The default scripting language

Authors should specify the default scripting language for all scripts in a document by including the following `META` declaration in the `HEAD`:

```
<META http-equiv="Content-Script-Type" content="type" >
```

where "type" is a content type [p.53] naming the scripting language. Examples of values include "text/tcl", "text/javascript", "text/vbscript".

In the absence of a `META` declaration, the default can be set by a "Content-Script-Type" HTTP header.

```
Content-Script-Type: type
```

where "type" is again a content type [p.53] naming the scripting language.

User agents should determine the default scripting language for a document according to the following steps (highest to lowest priority):

1. If any `META` declarations specify the "Content-Script-Type", the last one in the character stream determines the default scripting language.
2. Otherwise, if any HTTP headers specify the "Content-Script-Type", the last one in the character stream determines the default scripting language.

Documents that do not specify default scripting language information and that contain elements that specify an intrinsic event [p.254] script are incorrect. User agents may still attempt to interpret incorrectly specified scripts but are not required to. Authoring tools should generate default scripting language information to help authors avoid creating incorrect documents.

Local declaration of a scripting language

The `type` attribute must be specified for each `SCRIPT` element instance in a document. The value of the `type` attribute for a `SCRIPT` element overrides the default scripting language for that element.

In this example, we declare the default scripting language to be "text/tcl". We include one `SCRIPT` in the header, whose script is located in an external file and is in the scripting language "text/vbscript". We also include one `SCRIPT` in the body, which contains its own script written in "text/javascript".

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01//EN"
"http://www.w3.org/TR/html4/strict.dtd">
<HTML>
<HEAD>
<TITLE>A document with SCRIPT</TITLE>
<META http-equiv="Content-Script-Type" content="text/tcl">
<SCRIPT type="text/vbscript" src="http://someplace.com/progs/vbcalc">
</SCRIPT>
</HEAD>
<BODY>
<SCRIPT type="text/javascript">
...some JavaScript...
</SCRIPT>
</BODY>
</HTML>
```

References to HTML elements from a script

Each scripting language has its own conventions for referring to HTML objects from within a script. This specification does not define a standard mechanism for referring to HTML objects.

However, scripts should refer to an element according to its assigned name.

Scripting engines should observe the following precedence rules when identifying an element: a `name` attribute takes precedence over an `id` if both are set. Otherwise, one or the other may be used.

18.2.3 Intrinsic events

Note. *Authors of HTML documents are advised that changes are likely to occur in the realm of intrinsic events (e.g., how scripts are bound to events). Research in this realm is carried on by members of the W3C Document Object Model Working Group (see the W3C Web Site at <http://www.w3.org/> for more information).*

Attribute definitions

```
onload = script [p.57] [CT] [p.49]
```

The `onload` event occurs when the user agent finishes loading a window or all frames within a `FRAMESET`. This attribute may be used with `BODY` and `FRAMESET` elements.

`onunload = script [p.57] [CT] [p.49]`

The `onunload` event occurs when the user agent removes a document from a window or frame. This attribute may be used with `BODY` and `FRAMESET` elements.

`onclick = script [p.57] [CT] [p.49]`

The `onclick` event occurs when the pointing device button is clicked over an element. This attribute may be used with most elements.

`ondblclick = script [p.57] [CT] [p.49]`

The `ondblclick` event occurs when the pointing device button is double clicked over an element. This attribute may be used with most elements.

`onmousedown = script [p.57] [CT] [p.49]`

The `onmousedown` event occurs when the pointing device button is pressed over an element. This attribute may be used with most elements.

`onmouseup = script [p.57] [CT] [p.49]`

The `onmouseup` event occurs when the pointing device button is released over an element. This attribute may be used with most elements.

`onmouseover = script [p.57] [CT] [p.49]`

The `onmouseover` event occurs when the pointing device is moved onto an element. This attribute may be used with most elements.

`onmousemove = script [p.57] [CT] [p.49]`

The `onmousemove` event occurs when the pointing device is moved while it is over an element. This attribute may be used with most elements.

`onmouseout = script [p.57] [CT] [p.49]`

The `onmouseout` event occurs when the pointing device is moved away from an element. This attribute may be used with most elements.

`onfocus = script [p.57] [CT] [p.49]`

The `onfocus` event occurs when an element receives focus either by the pointing device or by tabbing navigation. This attribute may be used with the following elements: `A`, `AREA`, `LABEL`, `INPUT`, `SELECT`, `TEXTAREA`, and `BUTTON`.
`onblur = script [p.57] [CT] [p.49]`

The `onblur` event occurs when an element loses focus either by the pointing device or by tabbing navigation. It may be used with the same elements as `onfocus`.

`onkeypress = script [p.57] [CT] [p.49]`

The `onkeypress` event occurs when a key is pressed and released over an element. This attribute may be used with most elements.

`onkeydown = script [p.57] [CT] [p.49]`

The `onkeydown` event occurs when a key is pressed down over an element. This attribute may be used with most elements.

`onkeyup = script [p.57] [CT] [p.49]`

The `onkeyup` event occurs when a key is released over an element. This attribute may be used with most elements.

`onsubmit = script [p.57] [CT] [p.49]`

The `onsubmit` event occurs when a form is submitted. It only applies to the `FORM` element.

`onreset = script [p.57] [CT] [p.49]`

The `onreset` event occurs when a form is reset. It only applies to the `FORM` element.

`onselect = script [p.57] [CT] [p.49]`

The `onselect` event occurs when a user selects some text in a text field. This attribute may be used with the `INPUT` and `TEXTAREA` elements.

`onchange = script [p.57] [CT] [p.49]`

The `onchange` event occurs when a control loses the input focus and its value has been modified since gaining focus. This attribute applies to the following elements: `INPUT`, `SELECT`, and `TEXTAREA`.

It is possible to associate an action with a certain number of events that occur when a user interacts with a user agent. Each of the "intrinsic events" listed above takes a value that is a script. The script is executed whenever the event occurs for that element. The syntax of script data [p.57] depends on the scripting language.

Control elements such as `INPUT`, `SELECT`, `BUTTON`, `TEXTAREA`, and `LABEL` all respond to certain intrinsic events. When these elements do not appear within a form, they may be used to augment the graphical user interface of the document.

For instance, authors may want to include press buttons in their documents that do not submit a form but still communicate with a server when they are activated. The following examples show some possible control and user interface behavior based on intrinsic events.

In the following example, `userName` is a required text field. When a user attempts to leave the field, the `onblur` event calls a JavaScript function to confirm that `userName` has an acceptable value.

```
<INPUT NAME="userName" onBlur="validUserName(this.value)">
<INPUT NAME="num"
  onChange="if (!checkNum(this.value, 1, 10))
    {this.focus();this.select();} else {thanks()}"
  VALUE="0">
```

Here is another JavaScript example:

Here is a VBScript example of an event handler for a text field:

```
<INPUT name="edit1" size="50">
<SCRIPT type="text/vbscript">
  Sub edit1_changed()
    If edit1.value = "abc" Then
      button1.enabled = True
    Else
      button1.enabled = False
    End If
  End Sub
</SCRIPT>
```


Here is the same example using Tcl:

```
<INPUT name="edit1" size="50">
<SCRIPT type="text/tcl">
  proc edit1_changed {} {
    if {[edit value] == abc} {
      button1 enable 1
    } else {
      button1 enable 0
    }
  }
  edit1 onChange edit1_changed
</SCRIPT>
```

Here is a JavaScript example for event binding within a script. First, here's a simple click handler:

```
<BUTTON type="button" name="mybutton" value="10">
<SCRIPT type="text/javascript">
  function my_onclick() {
    . . .
  }
  document.form.mybutton.onclick = my_onclick
</SCRIPT>
</BUTTON>
```

Here's a more interesting window handler:

```
<SCRIPT type="text/javascript">
  function my_onload() {
    . . .
  }
  var win = window.open("some/other/URI")
  if (win) win.onload = my_onload
</SCRIPT>
```

In Tcl this looks like:

```
<SCRIPT type="text/tcl">
  proc my_onload {} {
    . . .
  }
  set win [window open "some/other/URI"]
  if {$win != ""} {
    $win onload my_onload
  }
</SCRIPT>
```

Note that "document.write" or equivalent statements in intrinsic event handlers create and write to a new document rather than modifying the current one.

18.2.4 Dynamic modification of documents

Scripts that are executed when a document is loaded may be able to modify the document's contents dynamically. The ability to do so depends on the scripting language itself (e.g., the "document.write" statement in the HTML object model supported by some vendors).

The dynamic modification of a document may be modeled as follows:

1. All `SCRIPT` elements are evaluated in order as the document is loaded.
2. All script constructs within a given `SCRIPT` element that generate SGML CDATA are evaluated. Their combined generated text is inserted in the document in place of the `SCRIPT` element.
3. The generated CDATA is re-evaluated.

HTML documents are constrained to conform to the HTML DTD both before and after processing any `SCRIPT` elements.

The following example illustrates how scripts may modify a document dynamically. The following script:

```
<TITLE>Test Document</TITLE>
<SCRIPT type="text/javascript">
  document.write("<p><b>Hello World!</b>")
</SCRIPT>
```

Has the same effect as this HTML markup:

```
<TITLE>Test Document</TITLE>
<P><B>Hello World!</B>
```

18.3 Designing documents for user agents that don't support scripting

The following sections discuss how authors may create documents that work for user agents that don't support scripting.

18.3.1 The `NOSCRIPT` element

```
<!ELEMENT NOSCRIPT - - (%block?)+
-- alternate content container for non script-based rendering -->
<!ATTLIST NOSCRIPT
  %attrs;
  -- %coreattrs, %ll8n, %events --
>
```

Start tag: required, End tag: required

The `NOSCRIPT` element allows authors to provide alternate content when a script is not executed. The content of a `NOSCRIPT` element should only be rendered by a script-aware user agent in the following cases:

- The user agent is configured not to evaluate scripts.
- The user agent doesn't support a scripting language invoked by a SCRIPT element earlier in the document.

User agents that do not support client-side scripts must render this element's contents.

In the following example, a user agent that executes the SCRIPT will include some dynamically created data in the document. If the user agent doesn't support scripts, the user may still retrieve the data through a link.

```
<SCRIPT type="text/tcl">
...some Tcl script to insert data...
</SCRIPT>
<NOSCRIPT>
<P>Access the <A href="http://someplace.com/data">data.</A>
</NOSCRIPT>
```

18.3.2 Hiding script data from user agents

User agents that don't recognize the SCRIPT element will likely render that element's contents as text. Some scripting engines, including those for languages JavaScript, VBScript, and Tcl allow the script statements to be enclosed in an SGML comment. User agents that don't recognize the SCRIPT element will thus ignore the comment while smart scripting engines will understand that the script in comments should be executed.

Another solution to the problem is to keep scripts in external documents and refer to them with the src attribute.

Commenting scripts in JavaScript

The JavaScript engine allows the string "<i>!--</i>" to occur at the start of a SCRIPT element, and ignores further characters until the end of the line. JavaScript interprets "/*" as starting a comment extending to the end of the current line. This is needed to hide the string "<i>!--</i>" from the JavaScript parser.

```
<SCRIPT type="text/javascript">
<i>!-- to hide script contents from old browsers
function square(i) {
    document.write("The call passed ", i, " to the function.", "<BR>")
    return i * i
}
document.write("The function returned ", square(5), ".")
// end hiding contents from old browsers -->
</SCRIPT>
```

Commenting scripts in VBScript

In VBScript, a single quote character causes the rest of the current line to be treated as a comment. It can therefore be used to hide the string "<i>!--</i>" from VBScript, for instance:

```
<SCRIPT type="text/vbscript">
<i>!--
Sub foo()
...
End Sub
' -->
</SCRIPT>
```

Commenting scripts in TCL

In Tcl, the "#" character comments out the rest of the line:

```
<SCRIPT type="text/tcl">
<i>!-- to hide script contents from old browsers
proc square {i} {
    document write "The call passed $i to the function.<BR>"
    return [expr $i * $i]
}
document write "The function returned [square 5]. "
# end hiding contents from old browsers -->
</SCRIPT>
```

Note. Some browsers close comments on the first "<i>" character, so to hide script content from such browsers, you can transpose operands for relational and shift operators (e.g., use "y < x" rather than "x > y") or use scripting language-dependent escapes for "<i>">

19 SGML reference information for HTML

Contents

- 1. Document Validation 261
- 2. Sample SGML catalog 262

The following sections contain the formal SGML definition of HTML 4. It includes the SGML declaration [p.263], the Document Type Definition [p.265] (DTD), and the Character entity references [p.299], as well as a sample SGML catalog [p.262].

These files are also available in ASCII format as listed below:

Default DTD:

strict.dtd

Transitional DTD:

loose.dtd

Frameset DTD:

frameset.dtd

SGML declaration:

HTML4.decl

Entity definition files:

HTMLspecial.ent
 HTMLsymbol.ent
 HTMLlat1.ent

A sample catalog:

HTML4.cat

19.1 Document Validation

Many authors rely on a limited set of browsers to check on the documents they produce, assuming that if the browsers can render their documents they are valid. Unfortunately, this is a very ineffective means of verifying a document's validity precisely because browsers are designed to cope with invalid documents by rendering them as well as they can to avoid frustrating users.

For better validation, you should check your document against an SGML parser such as nsgmls (see [SP] [p.357]), to verify that HTML documents conform to the HTML 4 DTD. If the document type declaration [p.60] of your document includes a URI and your SGML parser supports this type of system identifier, it will get the DTD directly. Otherwise you can use the following sample SGML catalog. It assumes that the DTD has been saved as the file "strict.dtd" and that the entities are in the files "HTMLlat1.ent", "HTMLsymbol.ent" and "HTMLspecial.ent". In any case, make sure your SGML parser is capable of handling [ISO10646]. [p.353] See your validation tool documentation for further details.

Beware that such validation, although useful and highly recommended, does not guarantee that a document fully conforms to the HTML 4 specification. This is because an SGML parser relies solely on the given SGML DTD which does not express all aspects of a valid HTML 4 document. Specifically, an SGML parser ensures that the syntax, the structure, the list of elements, and their attributes are valid. But for instance, it cannot catch errors such as setting the width attribute of an IMG element to an invalid value (i.e., "foo" or "12.5"). Although the specification restricts the value for this attribute to an "integer representing a length in pixels," the DTD only defines it to be CDATA [p.50], which actually allows any value. Only a specialized program could capture the complete specification of HTML 4.

Nevertheless, this type of validation is still highly recommended since it permits the detection of a large set of errors that make documents invalid.

19.2 Sample SGML catalog

This catalog includes the override directive to ensure that processing software such as nsgmls uses public identifiers in preference to system identifiers. This means that users do not have to be connected to the Web when retrieving URI-based system identifiers.

OVERRIDE YES

```
PUBLIC "-//W3C//DTD HTML 4.01//EN" strict.dtd
PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN" loose.dtd
PUBLIC "-//W3C//DTD HTML 4.01 Frameset//EN" frameset.dtd
PUBLIC "-//W3C//ENTITIES Latin1//HTML" HTMLlat1.ent
PUBLIC "-//W3C//ENTITIES Special//EN//HTML" HTMLspecial.ent
PUBLIC "-//W3C//ENTITIES Symbols//EN//HTML" HTMLsymbol.ent
```

20 SGML Declaration of HTML 4

Note. The total number of codepoints allowed in the document character set of this SGML declaration includes the first 17 planes of [ISO 10646] [p.353] (17 times 65536). This limitation has been made because this number is limited to a length of 8 digits in the current version of the SGML standard. It does not imply any statement about the feasibility of a long-term restriction of characters in UCS to the first 17 planes. Chances are very high that the limitation to 8 digits in SGML will be removed before, and that this specification will be updated before, the first assignment of a character beyond the first 17 planes.

Note. Strictly speaking, ISO Registration Number 177 refers to the original state of [ISO 10646] [p.353] in 1993. Changes since 1993 have been the addition of characters and a one-time operation reallocating a large number of codepoints for Korean Hangul (Amendment 5). Revisions of the HTML 4 specification may update the reference to ISO 10646 to include additional changes.

20.1 SGML Declaration

```
<!SGML "ISO 8879:1986 (WWW)"
--
SGML Declaration for HyperText Markup Language version HTML 4

With support for the first 17 planes of ISO 10646 and
increased limits for tag and literal lengths etc.

--
CHARSET
BASESET "ISO Registration Number 177//CHARSET
ISO/IEC 10646-1:1993 UCS-4 with
implementation level 3//ESC 2/5 2/15 4/6"
DESCSET 0 9 UNUSED
9 2 9
11 2 UNUSED
13 1 13
14 18 UNUSED
32 95 32
127 1 UNUSED
128 32 UNUSED
160 55136 160
55296 2048 UNUSED -- SURROGATES --
57344 1056768 57344

CAPACITY
SGMLREF 150000
TOTALCAP 150000
GRPCAP 150000
ENTCAP 150000

SCOPE DOCUMENT
SYNTAX
SHUNCHAR CONTROLS 0 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16
17 18 19 20 21 22 23 24 25 26 27 28 29 30 31 127
BASESET "ISO 646IRV:1991//CHARSET
```

```
International Reference Version
(IRV)//ESC 2/8 4/2"
DESCSET 0 128 0

FUNCTION
RE 13
RS 10
SPACE 32
TAB SEPCCHAR 9

NAMING
LCNMSTRT ""
UCNMSTRT ""
LCNMCHAR ".-_:;"
UCNMCHAR ".-_:;"
NAMECASE GENERAL YES
ENTITY NO

DELIM
GENERAL SGMMLREF
SHORTREF SGMMLREF
SGMLREF
QUANTITY SGMMLREF 60
ATTENT 60 -- increased --
ATTSPLEN 65536 -- These are the largest values --
LITLEN 65536 -- permitted in the declaration --
NAMELEN 65536 -- Avoid fixed limits in actual --
FILLN 65536 -- implementations of HTML UA's --
TAGLVL 100
TAGLEN 65536
GRPGTCNT 150
GRPCNT 64
```

```
FEATURES
MINIMIZE
DATATAG NO
OMITTAG YES
RANK NO
SHORTTAG YES
LINK
SIMPLE NO
IMPLICIT NO
EXPLICIT NO
OTHER
CONCUR NO
SUBDOC NO
FORMAL YES
APPINFO NONE
>
```

21 Document Type Definition

<!--

This is HTML 4.01 Strict DTD, which excludes the presentation attributes and elements that W3C expects to phase out as support for style sheets matures. Authors should use the Strict DTD when possible, but may use the Transitional DTD when support for presentation attribute and elements is required.

HTML 4 includes mechanisms for style sheets, scripting, embedding objects, improved support for right to left and mixed direction text, and enhancements to forms for improved accessibility for people with disabilities.

Draft: \$Date: 1999/12/24 22:40:35 \$

Authors:

Dave Raggett <dsr@w3.org>
 Arnaud Le Hors <lehors@w3.org>
 Ian Jacobs <ij@w3.org>

Further information about HTML 4.01 is available at:

<http://www.w3.org/TR/1999/REC-html401-19991224>.

The HTML 4.01 specification includes additional syntactic constraints that cannot be expressed within the DTDs.

-->

<!--

Typical usage:

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01//EN"
"http://www.w3.org/TR/html4/strict.dtd">
<html>
<head>
...
</head>
<body>
...
</body>
</html>
```

The URI used as a system identifier with the public identifier allows the user agent to download the DTD and entity sets as needed.

The FPI for the Transitional HTML 4.01 DTD is:

```
"-//W3C//DTD HTML 4.01 Transitional//EN"
```

This version of the transitional DTD is:

```
http://www.w3.org/TR/1999/REC-html401-19991224/loose.dtd
```

If you are writing a document that includes frames, use the following FPI:

```
"-//W3C//DTD HTML 4.01 Frameset//EN"
```

This version of the frameset DTD is:

```
http://www.w3.org/TR/1999/REC-html401-19991224/frameset.dtd
```

Use the following (relative) URIs to refer to the DTDs and entity definitions of this specification:

```
"strict.dtd"
"loose.dtd"
"frameset.dtd"
"HTMLlat1.ent"
"HTMLsymbol.ent"
"HTMLspecial.ent"
```

-->

```
<!-- Imported Names ----->
<!-- Feature Switch for frameset documents -->
<!ENTITY % HTML.Frameset "IGNORE">
```

```
<!ENTITY % ContentType "CDATA"
-- media type, as per [RFC2045]
-->
```

```
<!ENTITY % ContentTypes "CDATA"
-- comma-separated list of media types, as per [RFC2045]
-->
```

```
<!ENTITY % Charset "CDATA"
-- a character encoding, as per [RFC2045]
-->
```

```
<!ENTITY % Charsets "CDATA"
-- a space-separated list of character encodings, as per [RFC2045]
-->
```

```
<!ENTITY % LanguageCode "NAME"
-- a language code, as per [RFC1766]
-->
```

```
<!ENTITY % Character "CDATA"
-- a single character from [ISO10646]
-->
```

```
<!ENTITY % LinkTypes "CDATA"
-- space-separated list of link types
-->
```

```
<!ENTITY % MediaDesc "CDATA"
-- single or comma-separated list of media descriptors
-->
```

```
<!ENTITY % URI "CDATA"
-- a Uniform Resource Identifier,
see [URI]
-->
```

```
<!ENTITY % Datetime "CDATA" -- date and time information. ISO date format -->
```

```
<!ENTITY % Script "CDATA" -- script expression -->
```

```
<!ENTITY % StyleSheet "CDATA" -- style sheet data -->
```

```
<!ENTITY % Text "CDATA">
```

```

<!-- Parameter Entities -->
<!ENTITY % head.misc "SCRIPT|STYLE|META|LINK|OBJECT" -- repeatable head elements -->
<!ENTITY % heading "H1|H2|H3|H4|H5|H6">
<!ENTITY % list "UL | OL">
<!ENTITY % preformatted "PRE">

<!-- Character mnemonic entities ----->
<!ENTITY % HTMLlat1 PUBLIC
    "-//W3C//ENTITIES Latin1//EN//HTML"
    "HTMLlat1.ent">
%HTMLlat1;

<!ENTITY % HTMLsymbol PUBLIC
    "-//W3C//ENTITIES Symbols//EN//HTML"
    "HTMLsymbol.ent">
%HTMLsymbol;

<!ENTITY % HTMLspecial PUBLIC
    "-//W3C//ENTITIES Special//EN//HTML"
    "HTMLspecial.ent">
%HTMLspecial;

<!-- Generic Attributes ----->
<!ENTITY % coreattrs
    ID
    #IMPLIED -- document-wide unique id --
    class
    CDATA
    #IMPLIED -- space-separated list of classes --
    style
    %StyleSheet;
    #IMPLIED -- associated style info --
    title
    %Text;
    #IMPLIED -- advisory title --"
>

<!ENTITY % il18n
    "lang
    (%LanguageCode; #IMPLIED -- language code --
    dir (ltr|rtl) #IMPLIED -- direction for weak/neutral text --"
>

<!ENTITY % events
    "onclick %Script;
    ondblclick %Script;
    #IMPLIED -- a pointer button was clicked --
    onmousedown %Script;
    #IMPLIED -- a pointer button was pressed down --
    onmouseup %Script;
    #IMPLIED -- a pointer button was released --
    onmouseover %Script;
    #IMPLIED -- a pointer was moved onto --
    onmousemove %Script;
    #IMPLIED -- a pointer was moved within --
    onmouseout %Script;
    #IMPLIED -- a pointer was moved away --
    onkeypress %Script;
    #IMPLIED -- a key was pressed and released --
    onkeydown %Script;
    #IMPLIED -- a key was pressed down --
    onkeyup %Script;
    #IMPLIED -- a key was released --"
>

<!-- Reserved Feature Switch -->
<!ENTITY % HTML.Reserved "IGNORE">

<!-- The following attributes are reserved for possible future use -->
<{ %HTML.Reserved; [
<!ENTITY % reserved
    "dataasc %URI;
    dataid CDATA
    #IMPLIED -- a single or tabular Data Source --
    dataformatas (plaintext|html) plaintext -- text or html --"
    ]
>

```

```

]]>
<!ENTITY % reserved "">
<!ENTITY % attrs "%coreattrs; %il18n; %events; ">

<!-- Text Markup ----->
<!ENTITY % fontStyle
    "TT | I | B | BIG | SMALL">
<!ENTITY % phrase "EM | STRONG | DFN | CODE |
    SAMP | KBD | VAR | CITE | ABBR | ACRONYM" >
<!ENTITY % special
    "A | IMG | OBJECT | BR | SCRIPT | MAP | Q | SUB | SUP | SPAN | BDO">
<!ENTITY % formctrl "INPUT | SELECT | TEXTAREA | LABEL | BUTTON">
<!-- %inline; covers inline or "text-level" elements -->
<!ENTITY % inline "#PCDATA | %fontstyle; | %phrase; | %special; | %formctrl; ">
<!ELEMENT (%fontstyle; %phrase;) -- (%inline;)*>
<!ATTLIST (%fontstyle; %phrase;)
    %attrs;
    -- %coreattrs, %il18n, %events --
>
<!ELEMENT (SUB|SUP) -- (%inline;)* -- subscript, superscript -->
<!ATTLIST (SUB|SUP)
    %attrs;
    -- %coreattrs, %il18n, %events --
>
<!ELEMENT SPAN -- (%inline;)* -- generic language/style container -->
<!ATTLIST SPAN
    %attrs;
    %reserved;
    -- reserved for possible future use --
>
<!ELEMENT BDO -- (%inline;)* -- Bidi over-ride -->
<!ATTLIST BDO
    %coreattrs;
    lang
    %LanguageCode; #IMPLIED -- language code --
    dir (ltr|rtl) #REQUIRED -- directionality --
>
<!ELEMENT BR -- O EMPTY
    #coreattrs;
    -- forced line break -->
<!ATTLIST BR
    %attrs;
    -- id, class, style, title --
>

<!-- HTML content models ----->
<!--
    HTML has two basic content models:
    %inline; character level elements and text strings
    %block; block-like elements e.g. paragraphs and lists
-->
<!ENTITY % block
    "P | %heading; | %list; | %preformatted; | DL | DIV | NOSCRIPT |
    BLOCKQUOTE | FORM | HR | TABLE | FIELDSET | ADDRESS">

```

```

<ENTITY % Flow "%Block; | %inline;">
<!-- Document Body ----->
<ELEMENT BODY O (%Block;|SCRIPT)+ +(INS|DEL) -- document body -->
<ATTLIST BODY
  %attrs;
  onload #IMPLIED -- the document has been loaded --
  onunload #IMPLIED -- the document has been removed --
>

<ELEMENT ADDRESS -- (%inline)* -- information on author -->
<ATTLIST ADDRESS
  %attrs;
>

<ELEMENT DIV -- (%flow)* -- generic language/style container -->
<ATTLIST DIV
  %attrs;
  %reserved;
>

<!-- The Anchor Element ----->
<ENTITY % Shape "(rect|circle|poly|default)">
<ENTITY % Coords "CDATA" -- comma-separated list of lengths -->
<ELEMENT A -- (%inline:)* -(A) -- anchor -->
<ATTLIST A
  %attrs;
  %Charset; #IMPLIED -- char encoding of linked resource --
  %ContentType; #IMPLIED -- advisory content type --
  name CDATA -- named link end --
  href %URI; #IMPLIED -- URI for linked resource --
  hreflang %LanguageCode; #IMPLIED -- language code --
  rel %LinkTypes; #IMPLIED -- forward link types --
  rev %LinkTypes; #IMPLIED -- reverse link types --
  %accesskey %Character; #IMPLIED -- accessibility key character --
  %Shape; rect -- for use with client-side image maps --
  %Coords; #IMPLIED -- for use with client-side image maps --
  tabindex NUMBER #IMPLIED -- position in tabbing order --
  %Script; #IMPLIED -- the element got the focus --
  onblur #IMPLIED -- the element lost the focus --
>

<!-- Client-side image maps ----->
<!-- These can be placed in the same document or grouped in a
separate document although this isn't yet widely supported -->
<ELEMENT MAP -- ((%block:)| AREA)+ -- client-side image map -->
<ATTLIST MAP
  %attrs;
  name CDATA #REQUIRED -- for reference by usemap --
>

<ELEMENT AREA - O EMPTY -- client-side image map area -->
<ATTLIST AREA
  %attrs;
  %Shape; rect -- controls interpretation of coords --
  %Coords; #IMPLIED -- comma-separated list of lengths --
  href %URI; #IMPLIED -- URI for linked resource --
  nohref #IMPLIED -- this region has no action --
  alt %Text; #REQUIRED -- short description --
>

```

```

tabindex NUMBER #IMPLIED -- position in tabbing order --
%Character; #IMPLIED -- accessibility key character --
onfocus %URI; #IMPLIED -- the element got the focus --
onblur %URI; #IMPLIED -- the element lost the focus --
>

<!-- The LINK Element ----->
<!-- Relationship values can be used in principle:

a) for document specific toolbars/menus when used
with the LINK element in document head e.g.
start, contents, previous, next, index, end, help
b) to link to a separate style sheet (rel=stylesheet)
c) to make a link to a script (rel=script)
d) by stylesheets to control how collections of
html nodes are rendered into printed documents
e) to make a link to a printable version of this document
e.g. a postscript or pdf version (rel=alternate media=print)
-->

<!-- ELEMENT LINK - O EMPTY -- a media-independent link -->
<ATTLIST LINK
  %attrs;
  %Charset; #IMPLIED -- char encoding of linked resource --
  href %URI; #IMPLIED -- URI for linked resource --
  hreflang %LanguageCode; #IMPLIED -- language code --
  type %ContentType; #IMPLIED -- advisory content type --
  rel %LinkTypes; #IMPLIED -- forward link types --
  rev %LinkTypes; #IMPLIED -- reverse link types --
  media %MediaDesc; #IMPLIED -- for rendering on these media --
>

<!-- Images ----->
<!-- Length defined in strict DTD for cellpadding/cellspacing -->
<ENTITY % Length "CDATA" -- n for pixels or nn% for percentage length -->
<ENTITY % MultiLength "CDATA" -- pixel, percentage, or relative -->
<[ [%HTML.Frameset; [
  <ENTITY % MultiLengths "CDATA" -- comma-separated list of MultiLength -->
]]>
<ENTITY % Pixels "CDATA" -- integer representing length in pixels -->

<!-- To avoid problems with text-only UAs as well as
to make image content understandable and navigable
to users of non-visual UAs, you need to provide
a description with ALT, and avoid server-side image maps -->
<ELEMENT IMG - O EMPTY -- Embedded image -->
<ATTLIST IMG
  %attrs;
  src %URI; #REQUIRED -- URI of image to embed --
  alt %Text; #REQUIRED -- short description --
  longdesc %URI; #IMPLIED -- link to long description
  (complements alt) --
  name CDATA #IMPLIED -- name of image for scripting --
  height %Length; #IMPLIED -- override height --
  width %Length; #IMPLIED -- override width --
  usemap %URI; #IMPLIED -- use client-side image map --
  ismap (ismap) #IMPLIED -- use server-side image map --
>

```

```

<!-- USEMAP points to a MAP element which may be in this document
or an external document, although the latter is not widely supported -->
<!------- OBJECT ----->
<!-- OBJECT is used to embed objects as part of HTML pages
PARAM elements should precede other content. SGML mixed content
model technicality precludes specifying this formally ...
-->

```

```

<ELEMENT OBJECT -- (PARAM | %flow:)*
-- generic embedded object -->
<ATTLIST OBJECT
  %attrs;
  declare #IMPLIED -- declare but don't instantiate flag --
  %URI; #IMPLIED -- identifies an implementation --
  %URI; #IMPLIED -- base URI for classid, data, archive--
  %URI; #IMPLIED -- reference to object's data --
  type %contentType; #IMPLIED -- content type for data --
  archive CDATA #IMPLIED -- content type for code --
  standby %Text; #IMPLIED -- space-separated list of URIs --
  height %Length; #IMPLIED -- message to show while loading --
  width %Length; #IMPLIED -- override width --
  usemap %URI; #IMPLIED -- use client-side image map --
  name CDATA #IMPLIED -- submit as part of form --
  tabindex NUMBER #IMPLIED -- position in tabbing order --
  %reserved; -- reserved for possible future use --
>

```

```

<ELEMENT PARAM - O EMPTY
<ATTLIST PARAM
  id ID #IMPLIED -- document-wide unique id --
  name CDATA #REQUIRED -- property name --
  value CDATA #IMPLIED -- property value --
  valueType (DATA|REF|OBJECT) DATA -- How to interpret value --
  type %contentType; #IMPLIED -- content type for value
  when valueType=ref --
  >

```

```

<!------- Horizontal Rule ----->
<ELEMENT HR - O EMPTY -- horizontal rule -->
<ATTLIST HR
  %attrs;
  -- %coreattrs, %il8n, %events --
>
<!------- Paragraphs ----->
<ELEMENT P - O (%inline:)*
  -- paragraph -->
<ATTLIST P
  %attrs;
  -- %coreattrs, %il8n, %events --
>

```

```

<!------- Headings ----->
<!-- There are six levels of headings from H1 (the most important)
to H6 (the least important).
-->
<ELEMENT (hheading:)* -- (%inline:)* -- heading -->
<ATTLIST (hheading:)*
  %attrs;
  -- %coreattrs, %il8n, %events --
>

```

```

>
<!------- Preformatted Text ----->
<!-- excludes markup for images and changes in font size -->
<ENTITY % pre.exclusion "IMG|OBJECT|BIG|SMALL|SUB|SUP">
<ELEMENT PRE - (%inline:)* - (%pre.exclusion:)* -- preformatted text -->
<ATTLIST PRE
  %attrs;
  -- %coreattrs, %il8n, %events --
>

```

```

<!------- Inline Quotes ----->
<ELEMENT Q - (%inline:)* -- short inline quotation -->
<ATTLIST Q
  %URI; #IMPLIED -- URI for source document or msg --
  cite %URI; #IMPLIED -- URI for source document or msg --
>
<!------- Block-like Quotes ----->
<ELEMENT BLOCKQUOTE - (%block:|SCRIPT)+ -- long quotation -->
<ATTLIST BLOCKQUOTE
  %attrs; #IMPLIED -- URI for source document or msg --
  cite %URI; #IMPLIED -- URI for source document or msg --
>
<!------- Inserted/Deleted Text ----->

```

```

<!-- INS/DEL are handled by inclusion on BODY -->
<ELEMENT (INS|DEL) - (%flow:)* -- inserted text, deleted text -->
<ATTLIST (INS|DEL)
  %attrs;
  cite %URI; #IMPLIED -- info on reason for change --
  datetime %Datetime; #IMPLIED -- date and time of change --
>

```

```

<!------- Lists ----->
<!-- definition lists - DT for term, DD for its definition -->
<ELEMENT DL - (DT|DD)+ -- definition list -->
<ATTLIST DL
  %attrs;
  -- %coreattrs, %il8n, %events --
>
<ELEMENT DT - O (%inline:)* -- definition term -->
<ELEMENT DD - O (%flow:)* -- definition description -->
<ATTLIST (DT|DD)
  %attrs;
  -- %coreattrs, %il8n, %events --
>
<ELEMENT OL - (LI)+ -- ordered list -->
<ATTLIST OL
  %attrs;
  -- %coreattrs, %il8n, %events --
>
<!-- Unordered Lists (UL) bullet styles -->
<ELEMENT UL - (LI)+ -- unordered list -->
<ATTLIST UL
  %attrs;
  -- %coreattrs, %il8n, %events --
>

```



```

<!-- LI - O (%flow;)*
<|ATTLIST LI
  %attrs;
>
<!-- Forms
<|ELEMENT FORM - - (%block;|SCRIPT)+ -(FORM) -- interactive form -->
<|ATTLIST FORM
  %attrs;
  #REQUIRED -- server-side form handler --
  GET -- HTTP method used to submit the form --
  %ContentType; "application/x-www-urlencoded"
  %ContentTypes; #IMPLIED -- list of MIME Types for file upload --
  name CDATA #IMPLIED -- name of form for scripting --
  onsubmit %Script; #IMPLIED -- the form was submitted --
  onreset %Script; #IMPLIED -- the form was reset --
  accept-charset %Charsets; #IMPLIED -- list of supported charsets --
>
<!-- Each label must not contain more than ONE field -->
<|ELEMENT LABEL - - (%inline;)* -(LABEL) -- form field label text -->
<|ATTLIST LABEL
  %attrs;
  IDREF #IMPLIED -- matches field ID value --
  %coreattrs, %i18n, %events --
  accesskey %Character; #IMPLIED -- accessibility key character --
  onfocus %Script; #IMPLIED -- the element got the focus --
  onblur %Script; #IMPLIED -- the element lost the focus --
>
<|ENTITY & InputType
  "(TEXT | PASSWORD | CHECKBOX |
  RADIO | SUBMIT | RESET |
  FILE | HIDDEN | IMAGE | BUTTON)"
>
<!-- attribute name required for all but submit and reset -->
<|ELEMENT INPUT - O EMPTY
<|ATTLIST INPUT
  %attrs;
  type %InputType; TEXT
  name CDATA #IMPLIED
  value CDATA #IMPLIED
  checked (checked) #IMPLIED
  disabled (disabled) #IMPLIED
  readonly (readonly) #IMPLIED
  size CDATA #IMPLIED
  maxlength NUMBER #IMPLIED
  src %URI; #IMPLIED
  alt CDATA #IMPLIED
  usemap %URI; #IMPLIED
  ismap NUMBER #IMPLIED
  tabindex NUMBER #IMPLIED
  accesskey %Character; #IMPLIED
  onfocus %Script; #IMPLIED
  onblur %Script; #IMPLIED
  onselect %Script; #IMPLIED
  onchange %Script; #IMPLIED
  accept %ContentTypes; #IMPLIED
  %reserved;
>
<|ELEMENT SELECT - - (OPTGROUP|OPTION)+ -- option selector -->
<|ATTLIST SELECT
  %attrs;
  -- list item -->
  -- %coreattrs, %i18n, %events --
  #IMPLIED -- field name --
  #IMPLIED -- rows visible --
  #IMPLIED -- default is single selection --
  #IMPLIED -- unavailable in this context --
  #IMPLIED -- position in tabbing order --
  onfocus %Script; #IMPLIED -- the element got the focus --
  onblur %Script; #IMPLIED -- the element lost the focus --
  onchange %Script; #IMPLIED -- the element value was changed --
  %reserved;
  >
<|ELEMENT OPTGROUP - - (OPTION)+ -- option group -->
<|ATTLIST OPTGROUP
  %attrs;
  disabled (disabled) #IMPLIED -- %coreattrs, %i18n, %events --
  label %Text; #REQUIRED -- for use in hierarchical menus --
  >
<|ELEMENT OPTION - O (#PCDATA)
<|ATTLIST OPTION
  %attrs;
  selected (selected) #IMPLIED -- %coreattrs, %i18n, %events --
  disabled (disabled) #IMPLIED -- unavailable in this context --
  label %Text; #IMPLIED -- for use in hierarchical menus --
  value CDATA #IMPLIED -- defaults to element content --
  >
<|ELEMENT TEXTAREA - - (#PCDATA)
<|ATTLIST TEXTAREA
  %attrs;
  name CDATA #IMPLIED
  rows NUMBER #REQUIRED
  cols NUMBER #REQUIRED
  disabled (disabled) #IMPLIED -- unavailable in this context --
  readonly (readonly) #IMPLIED
  tabindex NUMBER #IMPLIED
  accesskey %Character; #IMPLIED -- position in tabbing order --
  onfocus %Script; #IMPLIED -- accessibility key character --
  onblur %Script; #IMPLIED -- the element got the focus --
  onchange %Script; #IMPLIED -- the element lost the focus --
  %reserved;
  >
<!-- #PCDATA is to solve the mixed content problem,
per specification only whitespace is allowed there!
-->
<|ELEMENT FIELDSET - - (#PCDATA,LEGEND,(%flow;)* -- form control group -->
<|ATTLIST FIELDSET
  %attrs;
  >
<|ELEMENT LEGEND - - (%inline;)* -- fieldset legend -->
<|ATTLIST LEGEND
  %attrs;
  accesskey %Character; #IMPLIED -- %coreattrs, %i18n, %events --
  >
<|ELEMENT BUTTON - -
(%flow;)* -(A|%formctrl;|FORM|FIELDSET)
-- push button -->
<|ATTLIST BUTTON
  %attrs;
  -- %coreattrs, %i18n, %events --

```

```

name          #IMPLIED
CDATA        #IMPLIED
value         (button|submit|reset) submit -- sent to server when submitted --
type         (disabled) -- for use as form button --
disabled     #IMPLIED -- unavailable in this context --
tabindex     NUMBER #IMPLIED -- position in tabbing order --
accesskey   %Character; #IMPLIED -- accessibility key character --
onfocus    %Script; #IMPLIED -- the element got the focus --
onblur     %Script; #IMPLIED -- the element lost the focus --
$reserved;  -- reserved for possible future use --
>
<!-- ===== Tables =====>
<!-- IETF HTML table standard, see [RFC1942] -->
<!--
The BORDER attribute sets the thickness of the frame around the
table. The default units are screen pixels.

The FRAME attribute specifies which parts of the frame around
the table should be rendered. The values are not the same as
CAPS to avoid a name clash with the ALIGN attribute.

The value "border" is included for backwards compatibility with
<TABLE BORDER> which yields frame=border and border=implied
For <TABLE BORDER=1> you get border=1 and frame=implied. In this
case, it is appropriate to treat this as frame=border for backwards
compatibility with deployed browsers.
-->
<ENTITY % Tframe "(void|above|below|hsides|lhs|rhs|vsides|box|border)">

<!--
The RULES attribute defines which rules to draw between cells:
If RULES is absent then assume:
"none" if BORDER is absent or BORDER=0 otherwise "all"
-->
<ENTITY % TRules "(none | groups | rows | cols | all)">

<!-- horizontal placement of table relative to document -->
<ENTITY % TAlign "(left|center|right)">

<!-- horizontal alignment attributes for cell contents -->
<ENTITY % cellhalign
"align (left|center|right|justify|char) #IMPLIED
char %Character; #IMPLIED -- alignment char, e.g. char=':'. --
charoff %Length; #IMPLIED -- offset for alignment char --"
>

<!-- vertical alignment attributes for cell contents -->
<ENTITY % cellvalign
"valign (top|middle|bottom|baseline) #IMPLIED"
>

<ELEMENT TABLE --
(CAPTION?, (COL*|COLGROUP*), THEAD?, TFOOT?, TBODY+)*
-->
<ELEMENT CAPTION -- (%inline;)* -- table caption -->
<ELEMENT THEAD -- O (TR)+ -- table header -->
<ELEMENT TFOOT -- O (TR)+ -- table footer -->
<ELEMENT TBODY -- O (TR)+ -- table body -->
<ELEMENT COLGROUP -- O (COL)* -- table column group -->
<ELEMENT COL -- O EMPTY -- table column -->
<ELEMENT TR -- O (TH|TD)+ -- table row -->
<ELEMENT TH|TD -- O (%Flow;)* -- table header cell, table data cell -->

```

```

<ATTLIST TABLE
  %attrs;
summary      #IMPLIED -- purpose/structure for speech output--
width       table width --
border      #IMPLIED -- controls frame width around table --
frame       #IMPLIED -- which parts of frame to render --
rules       #IMPLIED -- rulings between rows and cols --
cellspacing %Length; #IMPLIED -- spacing between cells --
cellpadding %Length; #IMPLIED -- spacing within cells --
$reserved;  -- reserved for possible future use --
datapagesize CDATA #IMPLIED -- reserved for possible future use --
>

<ATTLIST CAPTION
  %attrs;
>

<!--
COLGROUP groups a set of COL elements. It allows you to group
several semantically related columns together.
-->
<ATTLIST COLGROUP
  %attrs;
span        NUMBER 1 -- %coreattrs, %i18n, %events --
width       %MultiLength; #IMPLIED -- default width for enclosed COLS --
%cellhalign; #IMPLIED -- horizontal alignment in cells --
%cellvalign; #IMPLIED -- vertical alignment in cells --
>

<!--
COL elements define the alignment properties for cells in
one or more columns.

The WIDTH attribute specifies the width of the columns, e.g.

width=64      width in screen pixels
width=0.5*    relative width of 0.5

The SPAN attribute causes the attributes of one
COL element to apply to more than one column.
-->
<ATTLIST COL
  %attrs;
span        NUMBER 1 -- %coreattrs, %i18n, %events --
width       %MultiLength; #IMPLIED -- column width specification --
%cellhalign; #IMPLIED -- horizontal alignment in cells --
%cellvalign; #IMPLIED -- vertical alignment in cells --
>

<!--
Use THEAD to duplicate headers when breaking table
across page boundaries, or for static headers when
TBODY sections are rendered in scrolling panel.

Use TFOOT to duplicate footers when breaking table
across page boundaries, or for static footers when
TBODY sections are rendered in scrolling panel.

Use multiple TBODY sections when rules are needed
between groups of table rows.
-->
<ATTLIST (THEAD|TBODY|TFOOT)
  %attrs;
-- %coreattrs, %i18n, %events --

```

HTML 4 Document Type Definition

```

%cellhalign;
%cellvalign;
>
</ATTLIST TR
  %attrs;
  axis CDATA #IMPLIED
  headers IDREFS #IMPLIED
  scope %Scope; #IMPLIED
  rowspan NUMBER 1
  colspan NUMBER 1
%cellhalign;
%cellvalign;
>

<!-- Scope is simpler than headers attribute for common tables -->
<ENTITY % Scope "(row|col|rowgroup|colgroup)">

<!-- TH is for headers, TD for data, but for cells acting as both use TD -->
</ATTLIST (TH|TD)
  %attrs;
  abbr %Text; #IMPLIED
  axis CDATA #IMPLIED
  headers IDREFS #IMPLIED
  scope %Scope; #IMPLIED
  rowspan NUMBER 1
  colspan NUMBER 1
%cellhalign;
%cellvalign;
>

<!-- ===== Document Head =====>
<!-- %head.misc; defined earlier on as "SCRIPT|STYLE|META|LINK|OBJECT" -->
<ENTITY % head.content "TITLE & BASE?">

<ELEMENT HEAD O O (%head.content;)+(%head.misc;)-- document head -->
</ATTLIST HEAD
  %l18n;
  profile %URI; #IMPLIED -- named dictionary of meta info --
  >

<!-- The TITLE element is not considered part of the flow of text.
It should be displayed, for example as the page header or
window title. Exactly one title is required per document.
-->
<ELEMENT TITLE - - (#PCDATA) -(%head.misc;) -- document title -->
</ATTLIST TITLE %l18n>

<ELEMENT BASE - O EMPTY
  >
</ATTLIST BASE
  href %URI; #REQUIRED -- URI that acts as base URI --
  >

<ELEMENT META - O EMPTY
  >
</ATTLIST META
  http-equiv NAME #IMPLIED -- lang, dir, for use with content --
  name NAME #IMPLIED -- HTTP response header name --
  content CDATA #REQUIRED -- metainformation name --
  scheme CDATA #IMPLIED -- associated information --
  >

<ELEMENT STYLE - - %StyleSheet
  >
</ATTLIST STYLE
  %l18n;
  type %ContentType; #REQUIRED -- lang, dir, for use with title --
  >

```

HTML 4 Document Type Definition

```

media %MediaDesc; #IMPLIED -- designed for use with these media --
title %Text; #IMPLIED -- advisory title --
>

<ELEMENT SCRIPT - - %Script;
  >
</ATTLIST SCRIPT
  charset %Charset; #IMPLIED -- char encoding of linked resource --
  type %ContentType; #REQUIRED -- content type of script language --
  src %URI; #IMPLIED -- URI for an external script --
  defer (defer) #IMPLIED -- UA may defer execution of script --
  event CDATA #IMPLIED -- reserved for possible future use --
  for %URI; #IMPLIED -- reserved for possible future use --
  >

<ELEMENT NOSCRIPT - - (%block;)+
  >
</ATTLIST NOSCRIPT
  %attrs;
  >

<!-- ===== Document Structure =====>
<ENTITY % html.content "HEAD, BODY">

<ELEMENT HTML O O (%html.content;) -- document root element -->
</ATTLIST HTML
  %l18n;
  >
  -- lang, dir --

```



```

"lang          $LanguageCode: #IMPLIED -- language code --
dir           (ltr|rtl)          #IMPLIED -- direction for weak/neutral text ---"
>
<!ENTITY % events
"onclick     $Script;          #IMPLIED -- a pointer button was clicked --
ondblclick   $Script;          #IMPLIED -- a pointer button was double clicked--
onmousedown  $Script;          #IMPLIED -- a pointer button was pressed down --
onmouseup    $Script;          #IMPLIED -- a pointer button was released --
onmouseover  $Script;          #IMPLIED -- a pointer was moved onto --
onmouseout   $Script;          #IMPLIED -- a pointer was moved within --
onmousemove  $Script;          #IMPLIED -- a pointer was moved away --
onkeypress   $Script;          #IMPLIED -- a key was pressed and released --
onkeydown    $Script;          #IMPLIED -- a key was pressed down --
onkeyup      $Script;          #IMPLIED -- a key was released --"
>
<!-- Reserved Feature Switch -->
<!ENTITY % HTML.Reserved "IGNORE">
<!-- The following attributes are reserved for possible future use -->
<![ %HTML.Reserved; [
<!ENTITY % reserved
"datasc     $URI;             #IMPLIED -- a single or tabular Data Source --
datafid     CDATA             #IMPLIED -- the property or column name --
dataformatas (plaintext|html) plaintext -- text or html ---"
]]>
<!ENTITY % reserved "">
<!ENTITY % attrs "%coreattrs; #i18n; %events; ">
<!ENTITY % align "align (left|center|right|justify) #IMPLIED"
-- default is left for ltr paragraphs, right for rtl ---
>
<!--==== Text Markup =====>
<!ENTITY % fontstyle
"TT | I | B | U | S | STRIKE | BIG | SMALL">
<!ENTITY % phrase "EM | STRONG | DFN | CODE |
SAMP | KBD | VAR | CITE | ABBR | ACRONYM" >
<!ENTITY % special
"A | IMG | APPLET | OBJECT | FONT | BASEFONT | BR | SCRIPT |
MAP | Q | SUB | SUP | SPAN | BDO | IFRAME">
<!ENTITY % formctrl "INPUT | SELECT | TEXTAREA | LABEL | BUTTON">
<!-- $inline: covers inline or "text-level" elements -->
<!ENTITY % inline "#PCDATA | %fontstyle; | %phrase; | %special; | %formctrl;">
<!ELEMENT (%fontstyle;|%phrase;) - - (%inline;)*>
<!ATTLIST (%fontstyle;|%phrase;)
%attrs;
-- %coreattrs, #i18n, %events --
>
<!ELEMENT (SUB | SUP) - - (%inline;)* -- subscript, superscript ---
<!ATTLIST (SUB | SUP)
%attrs;
-- %coreattrs, #i18n, %events --
>
<!ELEMENT SPAN - - (%inline;)* -- generic language/style container -->

```

```

<!ENTITY % Frametarget "CDATA" -- render in this frame --->

```

```

<!ENTITY % Text "CDATA">

```

```

<!-- Parameter Entities -->

```

```

<!ENTITY % head.misc "SCRIPT|STYLE|META|LINK|OBJECT" -- repeatable head elements -->

```

```

<!ENTITY % heading "H1|H2|H3|H4|H5|H6">

```

```

<!ENTITY % list "UL | OL | DIR | MENU">

```

```

<!ENTITY % preformatted "PRE">

```

```

<!ENTITY % Color "CDATA" -- a color using SRGB: #RRGGBB as Hex values -->

```

```

<!-- There are also 16 widely known color names with their SRGB values:

```

```

Black = #000000   Green = #008000
Silver = #C0C0C0   Lime = #00FF00
Gray = #808080    Olive = #808000
White = #FFFFFF    Yellow = #FFFF00
Maroon = #800000   Navy = #000080
Red = #FF0000     Blue = #0000FF
Purple = #800080  Teal = #008080
Fuchsia = #FF00FF Aqua = #00FFFF
-->

```

```

<!ENTITY % bodycolors "
bgcolor      %Color;          #IMPLIED -- document background color --
text         %Color;          #IMPLIED -- document text color --
link         %Color;          #IMPLIED -- color of links --
vlink       %Color;          #IMPLIED -- color of visited links --
alink       %Color;          #IMPLIED -- color of selected links --"
>

```

```

<!--==== Character mnemonic entities =====>

```

```

<!ENTITY % HTMLlat1 PUBLIC
"--/W3C//ENTITIES Latin1/EN//HTML"
"HTMLlat1.ent">
%HTMLlat1;

```

```

<!ENTITY % HTMLsymbol PUBLIC
"--/W3C//ENTITIES Symbols/EN//HTML"
"HTMLsymbol.ent">
%HTMLsymbol;

```

```

<!ENTITY % HTMLspecial PUBLIC
"--/W3C//ENTITIES Special/EN//HTML"
"HTMLspecial.ent">
%HTMLspecial;

```

```

<!--==== Generic Attributes =====>

```

```

<!ENTITY % coreattrs
"id          ID
class       CDATA
style      %StyleSheet;
title      %Text;
-- document-wide unique id --
-- space-separated list of classes --
-- associated style info --
-- advisory title ---"
>

```

```

<!ENTITY % i18n

```

HTML 4 Transitional Document Type Definition

HTML 4 Transitional Document Type Definition

```

<ATTLIST SPAN
  %attrs;
  %reserved;
  >
  -- %coreattrs, %i18n, %events --
  -- reserved for possible future use --

<ELEMENT BDO - - (%inline;)*
  <ATTLIST BDO
    %coreattrs;
    lang
    dir
    (%ltr|rtl)
    #REQUIRED -- directionality --
  >
  -- I18N Bidi over-ride -->
  -- id, class, style, title --
  -- language code --
  -- languageCode; #IMPLIED
  -- #REQUIRED -- directionality --

<ELEMENT BASEFONT - O EMPTY
  <ATTLIST BASEFONT
    id
    size
    color
    face
    >
  #IMPLIED -- document-wide unique id --
  #REQUIRED -- base font size for FONT elements --
  #IMPLIED -- text color --
  #IMPLIED -- comma-separated list of font names --

<ELEMENT FONT - - (%inline;)*
  <ATTLIST FONT
    %i18n;
    size
    color
    face
    >
  -- lang, dir --
  #IMPLIED -- [+|-]nn e.g. size="+1", size="4" --
  #IMPLIED -- text color --
  #IMPLIED -- comma-separated list of font names --

  -- local change to font -->
  -- id, class, style, title --

<ELEMENT BR - O EMPTY
  <ATTLIST BR
    %coreattrs;
    clear
    >
  -- forced line break --
  -- id, class, style, title --
  -- (left|all|right|none) none -- control of text flow --

<!--
  HTML has two basic content models:
  -->
  %inline; character level elements and text strings
  %block; block-like elements e.g. paragraphs and lists

<ENTITY % block
  "P | %heading; | %list; | %preformatted; | DL | DIV | CENTER |
  NOSCRIPT | NOFRAMES | BLOCKQUOTE | FORM | INDEX | HR |
  TABLE | FIELDSET | ADDRESS">

<ENTITY % flow "%block; | %inline;";

<!-- ===== Document Body =====>
<ELEMENT BODY O O (%flow;)* + (INS|DEL) -- document body -->
<ATTLIST BODY
  %attrs;
  onload
  onunload
  background
  %bodycolors;
  >
  -- %coreattrs, %i18n, %events --
  -- the document has been loaded --
  -- the document has been removed --
  -- texture tile for document
  background --
  -- bgcolor, text, link, vlink, alink
  >

<ELEMENT ADDRESS - - ((%inline;)|P)* -- information on author -->
<ATTLIST ADDRESS
  %attrs;
  >
  -- %coreattrs, %i18n, %events --
  -- generic language/style container -->
  -- %coreattrs, %i18n, %events --
  -- align, text alignment --
  -- reserved for possible future use --

<ELEMENT CENTER - - (%flow;)*
  <ATTLIST CENTER
    %attrs;
    >
  -- shorthand for DIV align=center -->
  -- %coreattrs, %i18n, %events --

<!-- ===== The Anchor Element =====>
<ENTITY % Shape "(rect|circle|poly|default)">
<ENTITY % Coords "CDATA" -- comma-separated list of lengths -->

<ELEMENT A - - (%inline;)* - (A) -- anchor -->
<ATTLIST A
  %attrs;
  charset
  type
  name
  href
  hreflang
  target
  rel
  rev
  accesskey
  shape
  coords
  tabindex
  onfocus
  onblur
  >
  #IMPLIED -- char encoding of linked resource --
  #IMPLIED -- advisory content type --
  #IMPLIED -- named link end --
  #IMPLIED -- URI for linked resource --
  #IMPLIED -- language code --
  #IMPLIED -- renderer in this frame --
  #IMPLIED -- forward link types --
  #IMPLIED -- reverse link types --
  #IMPLIED -- accessibility key character --
  rect -- for use with client-side image maps --
  coords -- for use with client-side image maps --
  #IMPLIED -- position in tabbing order --
  #IMPLIED -- the element got the focus --
  #IMPLIED -- the element lost the focus --

<!-- ===== Client-side image maps =====>
<!-- These can be placed in the same document or grouped in a
  separate document although this isn't yet widely supported -->

<ELEMENT MAP - - ((%block; | AREA)+ -- client-side image map -->
<ATTLIST MAP
  %attrs;
  name
  CDATA
  #REQUIRED -- for reference by usemap --
  >

<ELEMENT AREA - O EMPTY
  <ATTLIST AREA
    %attrs;
    shape
    coords
    href
    target
    nohref
    alt
    tabindex
    accesskey
    onfocus
    onblur
    >
  -- controls interpretation of coords --
  -- comma-separated list of lengths --
  #IMPLIED -- URI for linked resource --
  #IMPLIED -- renderer in this frame --
  #IMPLIED -- this region has no action --
  #REQUIRED -- short description --
  alt --
  NUMBER -- position in tabbing order --
  #IMPLIED -- accessibility key character --
  #IMPLIED -- the element got the focus --
  #IMPLIED -- the element lost the focus --
  >

```

HTML 4 Transitional Document Type Definition

```

<!-- The LINK Element ----->
<!--
Relationship values can be used in principle:

a) for document specific toolbars/menus when used
with the LINK element in document head e.g.
start, contents, previous, next, index, end, help
b) to link to a separate style sheet (rel=stylesheet)
c) to make a link to a script (rel=script)
d) by stylesheets to control how collections of
html nodes are rendered into printed documents
e) to make a link to a printable version of this document
e.g. a postscript or pdf version (rel=alternate media=print)
-->
<!-->
<ELEMENT LINK - O EMPTY
<ATTLIST LINK
  %attrs;
  charset; #IMPLIED #IMPLIED -- char encoding of linked resource --
  href; #URI; #IMPLIED -- URI for linked resource --
  hreflang; #LanguageCode; #IMPLIED -- language code --
  type; #ContentType; #IMPLIED -- advisory content type --
  rel; #LinkTypes; #IMPLIED -- forward link types --
  rev; #LinkTypes; #IMPLIED -- reverse link types --
  media; #MediaDesc; #IMPLIED -- for rendering on these media --
  target; #FrameTarget; #IMPLIED -- render in this frame --
>
<!-- ===== Images =====>
<!-- Length defined in strict DTD for cellpadding/cellspacing -->
<ENTITY % Length "CDATA" -- nn for pixels or nn% for percentage length -->
<ENTITY % Multilength "CDATA" -- pixel, percentage, or relative -->
<!-- =====>
<!( %HTML,FrameSet; [
<ENTITY % Multilengths "CDATA" -- comma-separated list of Multilength -->
] ]>
<ENTITY % Pixels "CDATA" -- integer representing length in pixels -->
<ENTITY % IAlign "(top|middle|bottom|left|right)" -- center? -->
<!-- To avoid problems with text-only UAs as well as
to make image content understandable and navigable
to users of non-visual UAs, you need to provide
a description with ALT, and avoid server-side image maps -->
<ELEMENT IMG - O EMPTY
<ATTLIST IMG
  %attrs;
  src; #URI; #REQUIRED -- URI of image to embed --
  alt; #Text; #REQUIRED -- short description --
  longdesc; #URI; #IMPLIED -- link to long description
                                (complements alt) --
  name; #CDATA -- name of image for scripting --
  height; #IMPLIED -- override height --
  width; #IMPLIED -- override width --
  usemap; #URI; #IMPLIED -- use client-side image map --
  ismap; #IAlign; #IMPLIED -- use server-side image map --
  border; #Pixels; #IMPLIED -- vertical or horizontal alignment --
  hspace; #Pixels; #IMPLIED -- link border width --
  vspace; #Pixels; #IMPLIED -- horizontal gutter --
>

```

HTML 4 Transitional Document Type Definition

```

<!-- USEMAP points to a MAP element which may be in this document
or an external document, although the latter is not widely supported -->
<!-- ===== OBJECT =====>
<!--
OBJECT is used to embed objects as part of HTML pages
PARAM elements should precede other content. SGML mixed content
model technicality precludes specifying this formally ...
-->
<!-- ===== OBJECT =====>
<!-->
<ELEMENT OBJECT - (PARAM | %flow;)*
-- generic embedded object -->
<ATTLIST OBJECT
  %attrs;
  declare; #IMPLIED -- declare but don't instantiate flag --
  classid; #URI; #IMPLIED -- identifies an implementation --
  codebase; #URI; #IMPLIED -- base URI for classid, data, archive--
  data; #URI; #IMPLIED -- reference to object's data --
  type; #ContentType; #IMPLIED -- content type for data --
  codetype; #ContentType; #IMPLIED -- content type for code --
  archive; #CDATA -- space-separated list of URIs --
  standby; #Text; #IMPLIED -- message to show while loading --
  height; #Length; #IMPLIED -- override height --
  width; #Length; #IMPLIED -- override width --
  usemap; #URI; #IMPLIED -- use client-side image map --
  name; #CDATA -- submit as part of form --
  tabindex; #NUMBER -- position in tabbing order --
  align; #IAlign; #IMPLIED -- vertical or horizontal alignment --
  border; #Pixels; #IMPLIED -- link border width --
  hspace; #Pixels; #IMPLIED -- horizontal gutter --
  vspace; #Pixels; #IMPLIED -- vertical gutter --
  %reserved;
>
<!-- ===== Java APPLET =====>
<!-->
<ELEMENT PARAM - O EMPTY
<ATTLIST PARAM
  id; #IMPLIED -- document-wide unique id --
  name; #CDATA -- property name --
  value; #CDATA -- property value --
  valueType; (DATA|REF|OBJECT) DATA -- How to interpret value --
  type; #ContentType; #IMPLIED -- content type for value
                                when valueType=ref --
>
<!-- ===== Java APPLET =====>
<!-->
One of code or object attributes must be present.
Place PARAM elements before other content.
-->
<!-- ===== Java APPLET =====>
<!-->
<ELEMENT APPLET - (PARAM | %flow;)* -- Java applet -->
<ATTLIST APPLET
  %coreattrs;
  codebase; #URI; #IMPLIED -- optional base URI for applet --
  archive; #CDATA -- comma-separated archive list --
  code; #CDATA -- applet class file --
  object; #CDATA -- serialized applet file --
  alt; #Text; #IMPLIED -- short description --
  name; #CDATA -- allows applets to find each other --
  width; #Length; #REQUIRED -- initial width --
  height; #Length; #REQUIRED -- initial height --
  align; #IAlign; #IMPLIED -- vertical or horizontal alignment --
  hspace; #Pixels; #IMPLIED -- horizontal gutter --
  vspace; #Pixels; #IMPLIED -- vertical gutter --
>

```

```

<!-- Horizontal Rule =====>
<ELEMENT HR - O EMPTY -- horizontal rule -->
<ATTLIST HR
  %attrs;
  align (left|center|right) #IMPLIED
  noshade #IMPLIED
  size %Pixels; #IMPLIED
  width %Length; #IMPLIED
  >
<!-- Paragraphs =====>
<ELEMENT P - O (%inline;)* -- paragraph -->
<ATTLIST P
  %attrs;
  %align;
  >
<!-- Headings =====>
<!--
There are six levels of headings from H1 (the most important)
to H6 (the least important).
-->
<ELEMENT (%heading;) - - (%inline;)* -- heading -->
<ATTLIST (%heading;)
  %attrs;
  %align;
  >
<!-- Preformatted Text =====>
<!-- excludes markup for images and changes in font size -->
<ENTITY % pre.exclusion "IMG|OBJECT|APPLET|BIG|SMALL|SUB|SUP|FONT|BASEFONT">
<ELEMENT PRE - - (%inline;)* - (%pre.exclusion;) -- preformatted text -->
<ATTLIST PRE
  %attrs;
  width NUMBER #IMPLIED
  >
<!-- Inline Quotes =====>
<ELEMENT Q - - (%inline;)* -- short inline quotation -->
<ATTLIST Q
  %attrs;
  cite %URI; #IMPLIED -- URI for source document or msg --
  >
<!-- Block-like Quotes =====>
<ELEMENT BLOCKQUOTE - - (%flow;)* -- long quotation -->
<ATTLIST BLOCKQUOTE
  %attrs;
  cite %URI; #IMPLIED -- URI for source document or msg --
  >
<!-- Inserted/Deleted Text =====>
<!-- INS/DEL are handled by inclusion on BODY -->
<ELEMENT (INS|DEL) - - (%flow;)* -- inserted text, deleted text -->
<ATTLIST (INS|DEL)

```

```

  %attrs;
  cite %URI; #IMPLIED -- info on reason for change --
  datetime %Datetime; #IMPLIED -- date and time of change --
  >
<!-- Lists =====>
<!-- definition lists - DT for term, DD for its definition -->
<ELEMENT DL - - (DT|DD)+ -- definition list -->
<ATTLIST DL
  %attrs;
  compact (compact) #IMPLIED -- reduced interitem spacing --
  >
<ELEMENT DT - O (%inline;)* -- definition term -->
<ELEMENT DD - O (%flow;)* -- definition description -->
<ATTLIST (DT|DD)
  %attrs;
  >
<!-- Ordered lists (OL) Numbering style
1 arabic numbers 1, 2, 3, ...
a lower alpha a, b, c, ...
A upper alpha A, B, C, ...
i lower roman i, ii, iii, ...
I upper roman I, II, III, ...
The style is applied to the sequence number which by default
is reset to 1 for the first list item in an ordered list.
This can't be expressed directly in SGML due to case folding.
-->
<ENTITY % OLStyle "CDATA" -- constrained to: "(1|a|i|I)" -->
<ELEMENT OL - - (LI)+ -- ordered list -->
<ATTLIST OL
  %attrs;
  type %OLStyle; #IMPLIED -- numbering style --
  compact (compact) #IMPLIED -- reduced interitem spacing --
  start NUMBER #IMPLIED -- starting sequence number --
  >
<!-- Unordered Lists (UL) bullet styles -->
<ENTITY % ULStyle "(disc|square|circle)">
<ELEMENT UL - - (LI)+ -- unordered list -->
<ATTLIST UL
  %attrs;
  type %ULStyle; #IMPLIED -- bullet style --
  compact (compact) #IMPLIED -- reduced interitem spacing --
  >
<ELEMENT (DIR|MENU) - - (LI)+ - (%block;)* -- directory list, menu list -->
<ATTLIST (DIR|MENU)
  %attrs;
  compact (compact) #IMPLIED -- reduced interitem spacing --
  >
<ATTLIST MENU
  %attrs;
  compact (compact) #IMPLIED -- reduced interitem spacing --
  >

```



```

<!ENTITY % LIStyle "CDATA" -- constrained to: "(%LStyle:|%OLStyle:)" -->
<ELEMENT LI - O (%Flow)* -- list item -->
<ATTLIST LI
  %atts;
  type %LISyle; #IMPLIED
  value %NUMBER #IMPLIED
  >
<!-- ===== Forms ===== -->
<ELEMENT FORM - (%Flow)* -(FORM) -- interactive form -->
<ATTLIST FORM
  %atts;
  action %URI; #REQUIRED -- server-side form handler --
  method %GET|POST; #IMPLIED -- HTTP method used to submit the form--
  enctype %ContentType; "application/x-www-form-urlencoded"
  accept %ContentTypes; #IMPLIED -- list of MIME types for file upload --
  name %CDATA #IMPLIED -- name of form for scripting --
  onSubmit %Script; #IMPLIED -- the form was submitted --
  onReset %Script; #IMPLIED -- the form was reset --
  target %FrameTarget; #IMPLIED -- render in this frame --
  acceptCharset %Charsets; #IMPLIED -- list of supported charsets --
  >
<!-- Each label must not contain more than ONE field -->
<ELEMENT LABEL - (%Inline)* -(LABEL) -- form field label text -->
<ATTLIST LABEL
  %atts;
  for %IDREF #IMPLIED -- matches field ID value --
  accesskey %Character; #IMPLIED -- accessibility key character --
  onFocus %Script; #IMPLIED -- the element got the focus --
  onBlur %Script; #IMPLIED -- the element lost the focus --
  >
<ENTITY % InputType
  "(TEXT | PASSWORD | CHECKBOX |
  RADIO | SUBMIT | RESET |
  FILE | HIDDEN | IMAGE | BUTTON)"
  >
<!-- attribute name required for all but submit and reset -->
<ELEMENT INPUT - O EMPTY -- form control -->
<ATTLIST INPUT
  %atts;
  type %InputType; #IMPLIED -- what kind of widget is needed --
  name %CDATA #IMPLIED -- submit as part of form --
  value %CDATA #IMPLIED -- Specify for radio buttons and checkboxes --
  checked (%checked) #IMPLIED -- for radio buttons and check boxes --
  disabled (%disabled) #IMPLIED -- unavailable in this context --
  readOnly (%readOnly) #IMPLIED -- for text and passwd --
  size %CDATA #IMPLIED -- specific to each type of field --
  maxLength %NUMBER #IMPLIED -- max chars for text fields --
  src %URI; #IMPLIED -- for fields with images --
  alt %CDATA #IMPLIED -- short description --
  useMap %URI; #IMPLIED -- use client-side image map --
  isMap (%ismap) #IMPLIED -- use server-side image map --
  tabIndex %NUMBER #IMPLIED -- position in tabbing order --
  accesskey %Character; #IMPLIED -- accessibility key character --
  onFocus %Script; #IMPLIED -- the element got the focus --
  onBlur %Script; #IMPLIED -- the element lost the focus --
  onReset %Script; #IMPLIED -- some text was selected --
  onAccept %Script; #IMPLIED -- the element value was changed --
  accept %ContentTypes; #IMPLIED -- list of MIME types for file upload --
  align %Align; #IMPLIED -- vertical or horizontal alignment --
  %reserved;
  >

```

```

  >
  <!-- ELEMENT SELECT - (OPTGROUP|OPTION)+ -- option selector -->
  <ATTLIST SELECT
    %atts;
    name %CDATA #IMPLIED -- field name --
    size %NUMBER #IMPLIED -- rows visible --
    multiple (%multiple) #IMPLIED -- default is single selection --
    disabled (%disabled) #IMPLIED -- unavailable in this context --
    tabIndex %NUMBER #IMPLIED -- position in tabbing order --
    onFocus %Script; #IMPLIED -- the element got the focus --
    onBlur %Script; #IMPLIED -- the element lost the focus --
    onChange %Script; #IMPLIED -- the element value was changed --
    %reserved;
    >
  <!-- ELEMENT OPTGROUP - (OPTION)+ -- option group -->
  <ATTLIST OPTGROUP
    %atts;
    disabled (%disabled) #IMPLIED -- %coreattrs, %i18n, %events --
    label %Text; #REQUIRED -- for use in hierarchical menus --
    >
  <!-- ELEMENT OPTION - O (%PCDATA) -- selectable choice -->
  <ATTLIST OPTION
    %atts;
    selected (%selected) #IMPLIED -- %coreattrs, %i18n, %events --
    disabled (%disabled) #IMPLIED -- unavailable in this context --
    label %Text; #IMPLIED -- for use in hierarchical menus --
    value %CDATA #IMPLIED -- defaults to element content --
    >
  <!-- ELEMENT TEXTAREA - (%PCDATA) -- multi-line text field -->
  <ATTLIST TEXTAREA
    %atts;
    name %CDATA #IMPLIED -- %coreattrs, %i18n, %events --
    rows %NUMBER #REQUIRED -- %coreattrs, %i18n, %events --
    cols %NUMBER #REQUIRED -- %coreattrs, %i18n, %events --
    disabled (%disabled) #IMPLIED -- unavailable in this context --
    readOnly (%readOnly) #IMPLIED -- position in tabbing order --
    tabIndex %NUMBER #IMPLIED -- accessibility key character --
    onFocus %Script; #IMPLIED -- the element got the focus --
    onBlur %Script; #IMPLIED -- the element lost the focus --
    onselect %Script; #IMPLIED -- some text was selected --
    onChange %Script; #IMPLIED -- the element value was changed --
    %reserved;
    >
  <!-- #PCDATA is to solve the mixed content problem,
  per specification only whitespace is allowed there! -->
  <!-- ELEMENT FIELDSET - (%PCDATA,LEGEND,(%flow))* -- form control group -->
  <ATTLIST FIELDSET
    %atts;
    >
  <!-- ELEMENT LEGEND - (%Inline)* -- fieldset legend -->
  <ENTITY % LAlign "(top|bottom|left|right)">
  <ATTLIST LEGEND
    %atts;
    accesskey %Character; #IMPLIED -- %coreattrs, %i18n, %events --
    align %Align; #IMPLIED -- accessibility key character --
    >

```

HTML 4 Transitional Document Type Definition

```

>
<ELEMENT BUTTON - -
  (%flow;)* - (A|%formctrl;|FORM|ISINDEX|FIELDSET|IFRAME)
  -- push button -->
</ELEMENT BUTTON
  %attrs;
  name CDATA #IMPLIED
  value CDATA #IMPLIED
  type (button|submit|reset) submit -- for use as form button --
  disabled (disabled) #IMPLIED -- unavailable in this context --
  tabindex NUMBER #IMPLIED -- position in tabbing order --
  accesskey %Character; #IMPLIED -- accessibility key character --
  onfocus %Script; #IMPLIED -- the element got the focus --
  onblur %Script; #IMPLIED -- the element lost the focus --
  %reserved;
  >
<!-- ===== Tables =====>
<!-- IETF HTML table standard, see [RFC1942] -->
<!--
The BORDER attribute sets the thickness of the frame around the
table. The default units are screen pixels.

The FRAME attribute specifies which parts of the frame around
the table should be rendered. The values are not the same as
CAPS to avoid a name clash with the VALIGN attribute.

The value "border" is included for backwards compatibility with
<TABLE BORDER=1> you get border=1 and frame=implied. In this
case, it is appropriate to treat this as frame=border for backwards
compatibility with deployed browsers.
-->
<ENTITY % TFrame "(void|above|below|hsides|lhs|rhs|vsides|box|border)">
<!--
The RULES attribute defines which rules to draw between cells:
If RULES is absent then assume:
"none" if BORDER is absent or BORDER=0 otherwise "all"
-->
<ENTITY % TRules "(none | groups | rows | cols | all)">
<!-- horizontal placement of table relative to document -->
<ENTITY % TAlign "(left|center|right)">
<!-- horizontal alignment attributes for cell contents -->
<ENTITY % cellhalign
  "align (left|center|right|justify|char) #IMPLIED
  char %Character; #IMPLIED -- alignment char, e.g. char='.' --
  charoff %Length; #IMPLIED -- offset for alignment char --"
  >
<!-- vertical alignment attributes for cell contents -->
<ENTITY % cellvalign
  "valign (top|middle|bottom|baseline) #IMPLIED"
  >
<ELEMENT TABLE - -
  (CAPTION? (COL*|COLGROUP*)) TFRAME?, TFOOT?, TBODY+>
<ELEMENT CAPTION - - (%inline;)* -- table caption -->

```

HTML 4 Transitional Document Type Definition

```

<ELEMENT THEAD - - O (TR)+ -- table header -->
<ELEMENT TFOOT - - O (TR)+ -- table footer -->
<ELEMENT TBODY - - O (TR)+ -- table body -->
<ELEMENT COLGROUP - - O (COL)* -- table column group -->
<ELEMENT COL - - O EMPTY -- table column -->
<ELEMENT TR - - O (TH|TD)+ -- table row -->
<ELEMENT TH|TD - - O (%flow;)* -- table header cell, table data cell-->
</ELEMENT TABLE
  %attrs;
  %coreattrs, %i18n, %events --
  #IMPLIED -- purpose/structure for speech output--
  width %Length; #IMPLIED -- table width --
  border %Pixels; #IMPLIED -- controls frame width around table --
  frame %Frame; #IMPLIED -- which parts of frame to render --
  rules %Rules; #IMPLIED -- rulings between rows and cols --
  cellspacing %Length; #IMPLIED -- spacing between cells --
  cellpadding %Length; #IMPLIED -- spacing within cells --
  align %TAlign; #IMPLIED -- table position relative to window --
  bgcolor %Color; #IMPLIED -- background color for cells --
  datarowsize CDATA #IMPLIED -- reserved for possible future use --
  >
<ENTITY % CAlign "(top|bottom|left|right)">
</ELEMENT CAPTION
  %attrs;
  align %CAlign; #IMPLIED -- relative to table --
  >
<!--
COLGROUP groups a set of COL elements. It allows you to group
several semantically related columns together.
-->
</ELEMENT COLGROUP
  %attrs;
  span NUMBER 1 -- %coreattrs, %i18n, %events --
  width %MultiLength; #IMPLIED -- default number of columns in group --
  %cellhalign; -- default width for enclosed COLs --
  %cellvalign; -- horizontal alignment in cells --
  >
<!--
COL elements define the alignment properties for cells in
one or more columns.

The WIDTH attribute specifies the width of the columns, e.g.
width=64 width in screen pixels
width=0.5* relative width of 0.5

The SPAN attribute causes the attributes of one
COL element to apply to more than one column.
-->
</ELEMENT COL
  %attrs;
  span NUMBER 1 -- column groups and properties --
  width %MultiLength; #IMPLIED -- COL attributes affect N columns --
  %cellhalign; -- horizontal alignment in cells --
  %cellvalign; -- vertical alignment in cells --
  >
<!--
Use THEAD to duplicate headers when breaking table
across page boundaries, or for static headers when

```



```

%coreattrs;
%!l8n;
prompt %Text; #IMPLIED
-- id, class, style, title --
-- lang, dir --
-- prompt message -->

<ELEMENT BASE - O EMPTY
<ATTLIST BASE
href %URI; #IMPLIED -- URI that acts as base URI --
target %FrameTarget; #IMPLIED -- render in this frame --
>

<ELEMENT META - O EMPTY
-- generic metainformation -->
<ATTLIST META
%!l8n;
http-equiv NAME #IMPLIED -- lang, dir, for use with content --
name NAME #IMPLIED -- HTTP response header name --
content CDATA #REQUIRED -- metainformation name --
scheme CDATA #IMPLIED -- associated information --
-- select form of content --
>

<ELEMENT STYLE - - *StyleSheet
-- style info -->
<ATTLIST STYLE
%!l8n;
type %ContentType; #REQUIRED -- lang, dir, for use with title --
media %MediaDesc; #IMPLIED -- content type of style language --
title %Text; #IMPLIED -- designed for use with these media --
-- advisory title --
>

<ELEMENT SCRIPT - - *Script;
-- script statements -->
<ATTLIST SCRIPT
%Charset; #IMPLIED -- char encoding of linked resource --
type %ContentType; #REQUIRED -- content type of script language --
language CDATA #IMPLIED -- predefined script language name --
src %URI; #IMPLIED -- URI for an external script --
defer (defer) #IMPLIED -- UA may defer execution of script --
event CDATA #IMPLIED -- reserved for possible future use --
for %URI; #IMPLIED -- reserved for possible future use --
>

<ELEMENT NOSCRIPT - - (%Flow;)*
-- alternate content container for non script-based rendering -->
<ATTLIST NOSCRIPT
%attrs;
-- %coreattrs, %l18n, %events --
>

<!--===== Document Structure =====>
<ENTITY % version "version CDATA #FIXED '#HTML.Version;'">

<[ [HTML.Frameset; [
<ENTITY % html.content "HEAD, FRAMESET">
]]>

<ENTITY % html.content "HEAD, BODY">

<ELEMENT HTML O (%html.content; ) -- document root element -->
<ATTLIST HTML
%!l8n;
%version;
-- lang, dir --
>

```

23 Frameset Document Type Definition

```
<!--
```

```
This is the HTML 4.01 Frameset DTD, which should be
used for documents with frames. This DTD is identical
to the HTML 4.01 Transitional DTD except for the
content model of the "HTML" element: in frameset
documents, the "FRAMESET" element replaces the "BODY"
element.
```

```
Draft: $Date: 1999/12/24 23:20:08 $
```

```
Authors:
```

```
Dave Raggett <dsr@w3.org>
Arnaud Le Hors <lehors@w3.org>
Ian Jacobs <ij@w3.org>
```

```
Further information about HTML 4.01 is available at:
```

```
http://www.w3.org/TR/1999/REC-html401-19991224.
```

```
-->
<!ENTITY % HTML.Version "-//W3C//DTD HTML 4.01 Frameset//EN"
-- Typical usage:
```

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01 Frameset//EN"
"http://www.w3.org/TR/html4/frameset.dtd">
<html>
<head>
...
</head>
<frameset>
...
</frameset>
</html>
-->
```

```
<!ENTITY % HTML.Frameset "INCLUDE" >
<!ENTITY % HTML4.dtd PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN" >
%HTML4.dtd;
```

24 Character entity references in HTML 4

Contents

1. Introduction to character entity references	299
2. Character entity references for ISO 8859-1 characters	299
1. The list of characters	300
3. Character entity references for symbols, mathematical symbols, and Greek letters	303
1. The list of characters	304
4. Character entity references for markup-significant and internationalization characters	308
1. The list of characters	308

24.1 Introduction to character entity references

A character entity reference [p.45] is an SGML construct that references a character of the document character set. [p.41]

This version of HTML supports several sets of character entity references:

- ISO 8859-1 (Latin-1) characters [p.299] In accordance with section 14 of [RFC1866] [p.356], the set of Latin-1 entities has been extended by this specification to cover the whole right part of ISO-8859-1 (all code positions with the high-order bit set), including the already commonly used ` `, `©`, and `®`. The names of the entities are taken from the appendices of SGML (defined in [ISO8879] [p.353]).
- symbols, mathematical symbols, and Greek letters [p.303]. These characters may be represented by glyphs in the Adobe font "Symbol".
- markup-significant and internationalization characters [p.308] (e.g., for bidirectional text).

The following sections present the complete lists of character entity references. Although, by convention, [ISO10646] [p.353] the comments following each entry are usually written with uppercase letters, we have converted them to lowercase in this specification for reasons of readability.

24.2 Character entity references for ISO 8859-1 characters

The character entity references in this section produce characters whose numeric equivalents should already be supported by conforming HTML 2.0 user agents. Thus, the character entity reference `÷` is a more convenient form than `÷`; for obtaining the division sign (÷).

To support these named entities, user agents need only recognize the entity names and convert them to characters that lie within the repertoire of [ISO88591] [p.354].

Character 65533 (FFFF hexadecimal) is the last valid character in UCS-2. 65534 (FFFE hexadecimal) is unassigned and reserved as the byte-swapped version of ZERO WIDTH NON-BREAKING SPACE for byte-order detection purposes. 65535 (FFFF hexadecimal) is unassigned.

24.2.1 The list of characters

```
<!-- Portions © International Organization for Standardization 1986
Permission to copy in any form is granted for use with
conforming SGML systems and applications as defined in
ISO 8879, provided this notice is included in all copies.
-->
```

```
<!-- Character entity set. Typical invocation:
```

```
<!ENTITY % HTMLlat1 PUBLIC
    "-//W3C//ENTITIES Latin 1//EN/HTML">
%HTMLlat1;
```

```
-->
<!ENTITY nbsp CDATA "&#160;" -- no-break space = non-breaking space,
    U+00A0 ISONum -->
<!ENTITY iexcl CDATA "&#161;" -- inverted exclamation mark, U+00A1 ISONum -->
<!ENTITY cent CDATA "&#162;" -- cent sign, U+00A2 ISONum -->
<!ENTITY pound CDATA "&#163;" -- pound sign, U+00A3 ISONum -->
<!ENTITY pound CDATA "&#163;" -- pound sign, U+00A3 ISONum -->
<!ENTITY currn CDATA "&#164;" -- currency sign, U+00A4 ISONum -->
<!ENTITY yen CDATA "&#165;" -- yen sign = Yuan sign, U+00A5 ISONum -->
<!ENTITY brvbar CDATA "&#166;" -- broken bar = broken vertical bar,
    U+00A6 ISONum -->
<!ENTITY sect CDATA "&#167;" -- section sign, U+00A7 ISONum -->
<!ENTITY uml CDATA "&#168;" -- diaeresis = spacing diaeresis,
    U+00A8 ISODia -->
<!ENTITY copy CDATA "&#169;" -- copyright sign, U+00A9 ISONum -->
<!ENTITY ordf CDATA "&#170;" -- feminine ordinal indicator, U+00AA ISONum -->
<!ENTITY laquo CDATA "&#171;" -- left-pointing double angle quotation mark
    = left pointing guillemet, U+00AB ISONum -->
<!ENTITY not CDATA "&#172;" -- not sign, U+00AC ISONum -->
<!ENTITY shy CDATA "&#173;" -- soft hyphen = discretionary hyphen,
    U+00AD ISONum -->
<!ENTITY reg CDATA "&#174;" -- registered sign = registered trade mark sign,
    U+00AE ISONum -->
<!ENTITY macr CDATA "&#175;" -- macron = spacing macron = overline
    = APL overbar, U+00AF ISODia -->
<!ENTITY deg CDATA "&#176;" -- degree sign, U+00B0 ISONum -->
<!ENTITY plusmn CDATA "&#177;" -- plus-minus sign = plus-or-minus sign,
    U+00B1 ISONum -->
<!ENTITY sup2 CDATA "&#178;" -- superscript two = superscript digit two
    = squared, U+00B2 ISONum -->
<!ENTITY sup3 CDATA "&#179;" -- superscript three = superscript digit three
    = cubed, U+00B3 ISONum -->
<!ENTITY acute CDATA "&#180;" -- acute accent = spacing acute,
    U+00B4 ISODia -->
<!ENTITY micro CDATA "&#181;" -- micro sign, U+00B5 ISONum -->
<!ENTITY para CDATA "&#182;" -- pilcrow sign = paragraph sign,
    U+00B6 ISONum -->
<!ENTITY middot CDATA "&#183;" -- middle dot = Georgian comma
```

```

= greek middle dot, U+00B7 ISONUM -->
<!ENTITY cedil CDATA "&#184;" -- cedilla = spacing cedilla, U+00B8 ISODIA -->
<!ENTITY sup1 CDATA "&#185;" -- superscript one = superscript digit one,
U+00B9 ISONUM -->
<!ENTITY ordm CDATA "&#186;" -- masculine ordinal indicator,
U+00BA ISONUM -->
<!ENTITY raquo CDATA "&#187;" -- right-pointing double angle quotation mark
= right pointing guillemet, U+00BB ISONUM -->
<!ENTITY frac14 CDATA "&#188;" -- vulgar fraction one quarter
= fraction one quarter, U+00BC ISONUM -->
<!ENTITY frac12 CDATA "&#189;" -- vulgar fraction one half
= fraction one half, U+00BD ISONUM -->
<!ENTITY frac34 CDATA "&#190;" -- vulgar fraction three quarters
= fraction three quarters, U+00BE ISONUM -->
<!ENTITY iquest CDATA "&#191;" -- inverted question mark
= turned question mark, U+00BF ISONUM -->
<!ENTITY Agrave CDATA "&#192;" -- latin capital letter A with grave
= latin capital letter A grave,
U+00C0 ISOLAT1 -->
<!ENTITY Aacute CDATA "&#193;" -- latin capital letter A with acute,
U+00C1 ISOLAT1 -->
<!ENTITY Acirc CDATA "&#194;" -- latin capital letter A with circumflex,
U+00C2 ISOLAT1 -->
<!ENTITY Atilde CDATA "&#195;" -- latin capital letter A with tilde,
U+00C3 ISOLAT1 -->
<!ENTITY Auml CDATA "&#196;" -- latin capital letter A with diaeresis,
U+00C4 ISOLAT1 -->
<!ENTITY Aring CDATA "&#197;" -- latin capital letter A with ring above
= latin capital letter A ring,
U+00C5 ISOLAT1 -->
<!ENTITY AElig CDATA "&#198;" -- latin capital ligature AE,
U+00C6 ISOLAT1 -->
<!ENTITY Ccedil CDATA "&#199;" -- latin capital letter C with cedilla,
U+00C7 ISOLAT1 -->
<!ENTITY Egrave CDATA "&#200;" -- latin capital letter E with grave,
U+00C8 ISOLAT1 -->
<!ENTITY Eacute CDATA "&#201;" -- latin capital letter E with acute,
U+00C9 ISOLAT1 -->
<!ENTITY Ecirc CDATA "&#202;" -- latin capital letter E with circumflex,
U+00CA ISOLAT1 -->
<!ENTITY Euml CDATA "&#203;" -- latin capital letter E with diaeresis,
U+00CB ISOLAT1 -->
<!ENTITY Igrave CDATA "&#204;" -- latin capital letter I with grave,
U+00CC ISOLAT1 -->
<!ENTITY Iacute CDATA "&#205;" -- latin capital letter I with acute,
U+00CD ISOLAT1 -->
<!ENTITY Icirc CDATA "&#206;" -- latin capital letter I with circumflex,
U+00CE ISOLAT1 -->
<!ENTITY Iuml CDATA "&#207;" -- latin capital letter I with diaeresis,
U+00CF ISOLAT1 -->
<!ENTITY ETH CDATA "&#208;" -- latin capital letter ETH, U+00D0 ISOLAT1 -->
<!ENTITY Ntilde CDATA "&#209;" -- latin capital letter N with tilde,
U+00D1 ISOLAT1 -->
<!ENTITY Ograve CDATA "&#210;" -- latin capital letter O with grave,
U+00D2 ISOLAT1 -->
<!ENTITY Oacute CDATA "&#211;" -- latin capital letter O with acute,
U+00D3 ISOLAT1 -->
<!ENTITY Ocirc CDATA "&#212;" -- latin capital letter O with circumflex,
U+00D4 ISOLAT1 -->

```

```

<!ENTITY Otilde CDATA "&#213;" -- latin capital letter O with tilde,
U+00D5 ISOLAT1 -->
<!ENTITY Ouml CDATA "&#214;" -- latin capital letter O with diaeresis,
U+00D6 ISOLAT1 -->
<!ENTITY times CDATA "&#215;" -- multiplication sign, U+00D7 ISONUM -->
<!ENTITY Oslash CDATA "&#216;" -- latin capital letter O with stroke
= latin capital letter O slash,
U+00D8 ISOLAT1 -->
<!ENTITY Ugrave CDATA "&#217;" -- latin capital letter U with grave,
U+00D9 ISOLAT1 -->
<!ENTITY Uacute CDATA "&#218;" -- latin capital letter U with acute,
U+00DA ISOLAT1 -->
<!ENTITY Ucirc CDATA "&#219;" -- latin capital letter U with circumflex,
U+00DB ISOLAT1 -->
<!ENTITY Uuml CDATA "&#220;" -- latin capital letter U with diaeresis,
U+00DC ISOLAT1 -->
<!ENTITY Yacute CDATA "&#221;" -- latin capital letter Y with acute,
U+00DD ISOLAT1 -->
<!ENTITY THORN CDATA "&#222;" -- latin capital letter THORN,
U+00DE ISOLAT1 -->
<!ENTITY szlig CDATA "&#223;" -- latin small letter sharp s = ess-zed,
U+00DF ISOLAT1 -->
<!ENTITY agrave CDATA "&#224;" -- latin small letter a with grave
= latin small letter a grave,
U+00E0 ISOLAT1 -->
<!ENTITY aacute CDATA "&#225;" -- latin small letter a with acute,
U+00E1 ISOLAT1 -->
<!ENTITY acirc CDATA "&#226;" -- latin small letter a with circumflex,
U+00E2 ISOLAT1 -->
<!ENTITY atilde CDATA "&#227;" -- latin small letter a with tilde,
U+00E3 ISOLAT1 -->
<!ENTITY auml CDATA "&#228;" -- latin small letter a with diaeresis,
U+00E4 ISOLAT1 -->
<!ENTITY aring CDATA "&#229;" -- latin small letter a with ring above
= latin small letter a ring,
U+00E5 ISOLAT1 -->
<!ENTITY aelig CDATA "&#230;" -- latin small letter ae
= latin small ligature ae, U+00B6 ISOLAT1 -->
<!ENTITY ccedil CDATA "&#231;" -- latin small letter c with cedilla,
U+00E7 ISOLAT1 -->
<!ENTITY egrave CDATA "&#232;" -- latin small letter e with grave,
U+00E8 ISOLAT1 -->
<!ENTITY eacute CDATA "&#233;" -- latin small letter e with acute,
U+00E9 ISOLAT1 -->
<!ENTITY ecirc CDATA "&#234;" -- latin small letter e with circumflex,
U+00EA ISOLAT1 -->
<!ENTITY euml CDATA "&#235;" -- latin small letter e with diaeresis,
U+00EB ISOLAT1 -->
<!ENTITY igrave CDATA "&#236;" -- latin small letter i with grave,
U+00EC ISOLAT1 -->
<!ENTITY iacute CDATA "&#237;" -- latin small letter i with acute,
U+00ED ISOLAT1 -->
<!ENTITY icirc CDATA "&#238;" -- latin small letter i with circumflex,
U+00EE ISOLAT1 -->
<!ENTITY iuml CDATA "&#239;" -- latin small letter i with diaeresis,
U+00EF ISOLAT1 -->
<!ENTITY eth CDATA "&#240;" -- latin small letter eth, U+00F0 ISOLAT1 -->
<!ENTITY ntilde CDATA "&#241;" -- latin small letter n with tilde,
U+00F1 ISOLAT1 -->
<!ENTITY ograve CDATA "&#242;" -- latin small letter o with grave,

```

```

U+00F2 ISolat1 -->
<!ENTITY oacute CDATA "&#243;" -- latin small letter o with acute,
U+00F3 ISolat1 -->
<!ENTITY ocirc CDATA "&#244;" -- latin small letter o with circumflex,
U+00F4 ISolat1 -->
<!ENTITY otilde CDATA "&#245;" -- latin small letter o with tilde,
U+00F5 ISolat1 -->
<!ENTITY ouml CDATA "&#246;" -- latin small letter o with diaeresis,
U+00F6 ISolat1 -->
<!ENTITY divide CDATA "&#247;" -- division sign, U+00F7 ISOnum -->
<!ENTITY slash CDATA "&#248;" -- latin small letter o with stroke,
= latin small letter o slash,
U+00F8 ISolat1 -->
<!ENTITY ugrave CDATA "&#249;" -- latin small letter u with grave,
U+00F9 ISolat1 -->
<!ENTITY uacute CDATA "&#250;" -- latin small letter u with acute,
U+00FA ISolat1 -->
<!ENTITY ucirc CDATA "&#251;" -- latin small letter u with circumflex,
U+00FB ISolat1 -->
<!ENTITY uuml CDATA "&#252;" -- latin small letter u with diaeresis,
U+00FC ISolat1 -->
<!ENTITY yacute CDATA "&#253;" -- latin small letter y with acute,
U+00FD ISolat1 -->
<!ENTITY thorn CDATA "&#254;" -- latin small letter thorn,
U+00FE ISolat1 -->
<!ENTITY yuml CDATA "&#255;" -- latin small letter y with diaeresis,
U+00FF ISolat1 -->

```

24.3 Character entity references for symbols, mathematical symbols, and Greek letters

The character entity references in this section produce characters that may be represented by glyphs in the widely available Adobe Symbol font, including Greek characters, various bracketing symbols, and a selection of mathematical operators such as gradient, product, and summation symbols.

To support these entities, user agents may support full [ISO10646] [p.353] or use other means. Display of glyphs for these characters may be obtained by being able to display the relevant [ISO10646] [p.353] characters or by other means, such as internally mapping the listed entities, numeric character references, and characters to the appropriate position in some font that contains the requisite glyphs.

When to use Greek entities. *This entity set contains all the letters used in modern Greek. However, it does not include Greek punctuation, precomposed accented characters nor the non-spacing accents (tonos, dialytika) required to compose them. There are no archaic letters, Coptic-unique letters, or precomposed letters for Polytonic Greek. The entities defined here are not intended for the representation of modern Greek text and would not be an efficient representation; rather, they are intended for occasional Greek letters used in technical and mathematical works.*

24.3.1 The list of characters

```

<!-- Mathematical, Greek and Symbolic characters for HTML -->
<!-- Character entity set. Typical invocation:
<!ENTITY % HTMLSYMBOL PUBLIC
"-//W3C//ENTITIES Symbols//EN/HTML">
%HTMLsymbol; -->
<!-- Portions © International Organization for Standardization 1986:
permission to copy in any form is granted for use with
conforming SGML systems and applications as defined in
ISO 8879, provided this notice is included in all copies.
-->
<!-- Relevant ISO entity set is given unless names are newly introduced.
New names (i.e., not in ISO 8879 list) do not clash with any
existing ISO 8879 entity names. ISO 10646 character numbers
are given for each character, in hex. CDATA values are decimal
conversions of the ISO 10646 values and refer to the document
character set. Names are ISO 10646 names.
-->
<!-- Latin Extended-B -->
<!ENTITY lfnof CDATA "&#402;" -- latin small f with hook = function
= florn, U+0192 ISotech -->
<!-- Greek -->
<!ENTITY Alpha CDATA "&#913;" -- greek capital letter alpha, U+0391 -->
<!ENTITY Beta CDATA "&#914;" -- greek capital letter beta, U+0392 -->
<!ENTITY Gamma CDATA "&#915;" -- greek capital letter gamma,
U+0393 ISogrck3 -->
<!ENTITY Delta CDATA "&#916;" -- greek capital letter delta,
U+0394 ISogrck3 -->
<!ENTITY Epsilon CDATA "&#917;" -- greek capital letter epsilon, U+0395 -->
<!ENTITY Zeta CDATA "&#918;" -- greek capital letter zeta, U+0396 -->
<!ENTITY Eta CDATA "&#919;" -- greek capital letter eta, U+0397 -->
<!ENTITY Theta CDATA "&#920;" -- greek capital letter theta,
U+0398 ISogrck3 -->
<!ENTITY Iota CDATA "&#921;" -- greek capital letter iota, U+0399 -->
<!ENTITY Kappa CDATA "&#922;" -- greek capital letter kappa, U+039A -->
<!ENTITY Lambda CDATA "&#923;" -- greek capital letter lambda,
U+039B ISogrck3 -->
<!ENTITY Mu CDATA "&#924;" -- greek capital letter mu, U+039C -->
<!ENTITY Nu CDATA "&#925;" -- greek capital letter nu, U+039D -->
<!ENTITY Xi CDATA "&#926;" -- greek capital letter xi, U+039E ISogrck3 -->
<!ENTITY Omicron CDATA "&#927;" -- greek capital letter omicron, U+039F -->
<!ENTITY Pi CDATA "&#928;" -- greek capital letter pi, U+03A0 ISogrck3 -->
<!ENTITY Rho CDATA "&#929;" -- greek capital letter rho, U+03A1 -->
<!-- there is no SigmaI, and no U+03A2 character either -->
<!ENTITY Sigma CDATA "&#931;" -- greek capital letter sigma,
U+03A3 ISogrck3 -->
<!ENTITY Tau CDATA "&#932;" -- greek capital letter tau, U+03A4 -->
<!ENTITY Upsilon CDATA "&#933;" -- greek capital letter upsilon,
U+03A5 ISogrck3 -->
<!ENTITY Phi CDATA "&#934;" -- greek capital letter phi,
U+03A6 ISogrck3 -->
<!ENTITY Chi CDATA "&#935;" -- greek capital letter chi, U+03A7 -->

```



```

<!ENTITY Psi          CDATA "&#936;" -- greek capital letter psi,
U+03A8 ISOgrk3 -->
<!ENTITY Omega       CDATA "&#937;" -- greek capital letter omega,
U+03A9 ISOgrk3 -->
<!ENTITY alpha       CDATA "&#945;" -- greek small letter alpha,
U+03B1 ISOgrk3 -->
<!ENTITY beta        CDATA "&#946;" -- greek small letter beta, U+03B2 ISOgrk3 -->
<!ENTITY gamma       CDATA "&#947;" -- greek small letter gamma,
U+03B3 ISOgrk3 -->
<!ENTITY delta       CDATA "&#948;" -- greek small letter delta,
U+03B4 ISOgrk3 -->
<!ENTITY epsilon     CDATA "&#949;" -- greek small letter epsilon,
U+03B5 ISOgrk3 -->
<!ENTITY zeta        CDATA "&#950;" -- greek small letter zeta, U+03B6 ISOgrk3 -->
<!ENTITY eta         CDATA "&#951;" -- greek small letter eta, U+03B7 ISOgrk3 -->
<!ENTITY theta       CDATA "&#952;" -- greek small letter theta,
U+03B8 ISOgrk3 -->
<!ENTITY iota        CDATA "&#953;" -- greek small letter iota, U+03B9 ISOgrk3 -->
<!ENTITY kappa       CDATA "&#954;" -- greek small letter kappa,
U+03BA ISOgrk3 -->
<!ENTITY lambda      CDATA "&#955;" -- greek small letter lambda,
U+03BB ISOgrk3 -->
<!ENTITY mu          CDATA "&#956;" -- greek small letter mu, U+03BC ISOgrk3 -->
<!ENTITY nu          CDATA "&#957;" -- greek small letter nu, U+03BD ISOgrk3 -->
<!ENTITY xi          CDATA "&#958;" -- greek small letter xi, U+03BE ISOgrk3 -->
<!ENTITY omicron     CDATA "&#959;" -- greek small letter omicron, U+03BF NEW -->
<!ENTITY pi          CDATA "&#960;" -- greek small letter pi, U+03C0 ISOgrk3 -->
<!ENTITY rho         CDATA "&#961;" -- greek small letter rho, U+03C1 ISOgrk3 -->
<!ENTITY sigmaf      CDATA "&#962;" -- greek small letter final sigma,
U+03C2 ISOgrk3 -->
<!ENTITY sigma       CDATA "&#963;" -- greek small letter sigma,
U+03C3 ISOgrk3 -->
<!ENTITY tau         CDATA "&#964;" -- greek small letter tau, U+03C4 ISOgrk3 -->
<!ENTITY upsilon     CDATA "&#965;" -- greek small letter upsilon,
U+03C5 ISOgrk3 -->
<!ENTITY phi         CDATA "&#966;" -- greek small letter phi, U+03C6 ISOgrk3 -->
<!ENTITY chi         CDATA "&#967;" -- greek small letter chi, U+03C7 ISOgrk3 -->
<!ENTITY psi         CDATA "&#968;" -- greek small letter psi, U+03C8 ISOgrk3 -->
<!ENTITY omega       CDATA "&#969;" -- greek small letter omega,
U+03C9 ISOgrk3 -->
<!ENTITY thetasym    CDATA "&#977;" -- greek small letter theta symbol,
U+03D1 NEW -->
<!ENTITY upsih      CDATA "&#978;" -- greek upsilon with hook symbol,
U+03D2 NEW -->
<!ENTITY piv        CDATA "&#982;" -- greek pi symbol, U+03D6 ISOgrk3 -->
<!-- General Punctuation -->
<!ENTITY bull       CDATA "&#8226;" -- bullet = black small circle,
U+2022 ISOpub -->
<!-- bullet is NOT the same as bullet operator, U+2219 -->
<!ENTITY hellip     CDATA "&#8230;" -- horizontal ellipsis = three dot leader,
U+2026 ISOpub -->
<!ENTITY prime      CDATA "&#8242;" -- prime = minutes = feet, U+2032 ISOTech -->
<!ENTITY Prime      CDATA "&#8243;" -- double prime = seconds = inches,
U+2033 ISOTech -->
<!ENTITY oline      CDATA "&#8254;" -- overline = spacing overscore,
U+203E NEW -->
<!ENTITY frasl      CDATA "&#8260;" -- fraction slash, U+2044 NEW -->

```

```

<!-- Letterlike Symbols -->
<!ENTITY weierp     CDATA "&#8472;" -- script capital P = power set
= Weierstrass P, U+2118 ISOamsa -->
<!ENTITY image      CDATA "&#8465;" -- blackletter capital I = imaginary part,
U+2111 ISOamsa -->
<!ENTITY real       CDATA "&#8476;" -- blackletter capital R = real part symbol,
U+211C ISOamsa -->
<!ENTITY trade      CDATA "&#8482;" -- trade mark sign, U+2122 ISOnum -->
<!ENTITY alefsym    CDATA "&#8501;" -- alef symbol = first transfinite cardinal,
U+2135 NEW -->
<!-- alef symbol is NOT the same as hebrew letter alef,
U+05D0 although the same glyph could be used to depict both characters -->
<!-- Arrows -->
<!ENTITY larr       CDATA "&#8592;" -- leftwards arrow, U+2190 ISOnum -->
<!ENTITY uarr       CDATA "&#8593;" -- upwards arrow, U+2191 ISOnum -->
<!ENTITY rarr       CDATA "&#8594;" -- rightwards arrow, U+2192 ISOnum -->
<!ENTITY darr       CDATA "&#8595;" -- downwards arrow, U+2193 ISOnum -->
<!ENTITY harr       CDATA "&#8596;" -- left right arrow, U+2194 ISOamsa -->
<!ENTITY crarr      CDATA "&#8629;" -- downwards arrow with corner leftwards
= carriage return, U+21B5 NEW -->
<!ENTITY larr       CDATA "&#8656;" -- leftwards double arrow, U+21D0 ISOTech -->
<!-- ISO 10646 does not say that lArr is the same as the 'is implied by' arrow
but also does not have any other character for that function. So ? lArr can
be used for 'is implied by' as ISOTech suggests -->
<!ENTITY uArr       CDATA "&#8657;" -- upwards double arrow, U+21D1 ISOamsa -->
<!ENTITY rArr       CDATA "&#8658;" -- rightwards double arrow,
U+21D2 ISOTech -->
<!-- ISO 10646 does not say this is the 'implies' character but does not have
another character with this function so ?
rArr can be used for 'implies' as ISOTech suggests -->
<!ENTITY dArr       CDATA "&#8659;" -- downwards double arrow, U+21D3 ISOamsa -->
<!ENTITY hArr       CDATA "&#8660;" -- left right double arrow,
U+21D4 ISOamsa -->
<!-- Mathematical Operators -->
<!ENTITY forall     CDATA "&#8704;" -- for all, U+2200 ISOTech -->
<!ENTITY part       CDATA "&#8706;" -- partial differential, U+2202 ISOTech -->
<!ENTITY exist      CDATA "&#8707;" -- there exists, U+2203 ISOTech -->
<!ENTITY empty      CDATA "&#8709;" -- empty set = null set = diameter,
U+2205 ISOamsa -->
<!ENTITY nabla      CDATA "&#8711;" -- nabla = backward difference,
U+2207 ISOTech -->
<!ENTITY isin       CDATA "&#8712;" -- element of, U+2208 ISOTech -->
<!ENTITY notin      CDATA "&#8713;" -- not an element of, U+2209 ISOTech -->
<!ENTITY ni         CDATA "&#8715;" -- contains as member, U+220B ISOTech -->
<!-- should there be a more memorable name than 'ni'? -->
<!ENTITY prod       CDATA "&#8719;" -- n-ary product = product sign,
U+220F ISOamsa -->
<!-- prod is NOT the same character as U+03A0 'greek capital letter pi' though
the same glyph might be used for both -->
<!ENTITY sum        CDATA "&#8721;" -- n-ary sumation, U+2211 ISOamsb -->
<!-- sum is NOT the same character as U+03A3 'greek capital letter sigma'
though the same glyph might be used for both -->
<!ENTITY minus      CDATA "&#8722;" -- minus sign, U+2212 ISOTech -->
<!ENTITY lowast     CDATA "&#8727;" -- asterisk operator, U+2217 ISOTech -->
<!ENTITY radic      CDATA "&#8730;" -- square root = radical sign,
U+221A ISOTech -->
<!ENTITY prop       CDATA "&#8733;" -- proportional to, U+221D ISOTech -->
<!ENTITY infin      CDATA "&#8734;" -- infinity, U+221E ISOTech -->

```

```

<!ENTITY ang CDATA "&#8736;" -- angle, U+2220 ISOamsn -->
<!ENTITY and CDATA "&#8743;" -- logical and = wedge, U+2227 ISOTECH -->
<!ENTITY or CDATA "&#8744;" -- logical or = vee, U+2228 ISOTECH -->
<!ENTITY cap CDATA "&#8745;" -- intersection = cap, U+2229 ISOTECH -->
<!ENTITY cup CDATA "&#8746;" -- union = cup, U+222A ISOTECH -->
<!ENTITY int CDATA "&#8747;" -- integral, U+222B ISOTECH -->
<!ENTITY there4 CDATA "&#8756;" -- therefore, U+2234 ISOTECH -->
<!ENTITY slm CDATA "&#8764;" -- tilde operator = varies with = similar to,
U+223C ISOTECH -->
<!-- tilde operator is NOT the same character as the tilde, U+007E,
although the same glyph might be used to represent both -->
<!ENTITY cong CDATA "&#8773;" -- approximately equal to, U+2245 ISOTECH -->
<!ENTITY asymp CDATA "&#8776;" -- almost equal to = asymptotic to,
U+2248 ISOamsr -->
<!ENTITY ne CDATA "&#8800;" -- not equal to, U+2260 ISOTECH -->
<!ENTITY equiv CDATA "&#8801;" -- identical to, U+2261 ISOTECH -->
<!ENTITY le CDATA "&#8804;" -- less-than or equal to, U+2264 ISOTECH -->
<!ENTITY ge CDATA "&#8805;" -- greater-than or equal to,
U+2265 ISOTECH -->
<!ENTITY sub CDATA "&#8834;" -- subset of, U+2282 ISOTECH -->
<!ENTITY sup CDATA "&#8835;" -- superset of, U+2283 ISOTECH -->
<!-- note that nsup, 'not a superset of, U+2283' is not covered by the Symbol
font encoding and is not included. Should it be, for symmetry?
It is in ISOamsn -->
<!ENTITY nsub CDATA "&#8836;" -- not a subset of, U+2284 ISOamsn -->
<!ENTITY ssub CDATA "&#8838;" -- subset of or equal to, U+2286 ISOTECH -->
<!ENTITY supe CDATA "&#8839;" -- superset of or equal to,
U+2287 ISOTECH -->
<!ENTITY oplus CDATA "&#8853;" -- circled plus = direct sum,
U+2295 ISOamsb -->
<!ENTITY otimes CDATA "&#8855;" -- circled times = vector product,
U+2297 ISOamsb -->
<!ENTITY perp CDATA "&#8869;" -- up tack = orthogonal to = perpendicular,
U+22A5 ISOTECH -->
<!ENTITY sdot CDATA "&#8901;" -- dot operator, U+22C5 ISOamsb -->
<!-- dot operator is NOT the same character as U+00B7 middle dot -->
<!-- Miscellaneous Technical -->
<!ENTITY lceil CDATA "&#8968;" -- left ceiling = apl upstile,
U+2308 ISOamsc -->
<!ENTITY rceil CDATA "&#8969;" -- right ceiling, U+2309 ISOamsc -->
<!ENTITY lfloor CDATA "&#8970;" -- left floor = apl downstile,
U+230A ISOamsc -->
<!ENTITY rfloor CDATA "&#8971;" -- right floor, U+230B ISOamsc -->
<!ENTITY lang CDATA "&#9001;" -- left-pointing angle bracket = bra,
U+2329 ISOTECH -->
<!-- lang is NOT the same character as U+003C 'less than',
or U+2039 'single left-pointing angle quotation mark' -->
<!ENTITY rang CDATA "&#9002;" -- right-pointing angle bracket = ket,
U+232A ISOTECH -->
<!-- rang is NOT the same character as U+003E 'greater than',
or U+203A 'single right-pointing angle quotation mark' -->
<!-- Geometric Shapes -->
<!ENTITY loz CDATA "&#9674;" -- lozenge, U+25CA ISOpub -->
<!-- Miscellaneous Symbols -->
<!ENTITY spades CDATA "&#9824;" -- black spade suit, U+2660 ISOpub -->
<!-- black here seems to mean filled as opposed to hollow -->
<!ENTITY clubs CDATA "&#9827;" -- black club suit = shamrock,

```

```

U+2663 ISOpub -->
<!ENTITY hearts CDATA "&#9829;" -- black heart suit = valentine,
U+2665 ISOpub -->
<!ENTITY diams CDATA "&#9830;" -- black diamond suit, U+2666 ISOpub -->

```

24.4 Character entity references for markup-significant and internationalization characters

The character entity references in this section are for escaping markup-significant characters (these are the same as those in HTML 2.0 and 3.2), for denoting spaces and dashes. Other characters in this section apply to internationalization issues such as the disambiguation of bidirectional text (see the section on bidirectional text [p.82] for details).

Entities have also been added for the remaining characters occurring in CP-1252 which do not occur in the HTMLat1 or HTMLsymbol entity sets. These all occur in the 128 to 159 range within the CP-1252 charset. These entities permit the characters to be denoted in a platform-independent manner.

To support these entities, user agents may support full [ISO10646] [p.353] or use other means. Display of glyphs for these characters may be obtained by being able to display the relevant [ISO10646] [p.353] characters or by other means, such as internally mapping the listed entities, numeric character references, and characters to the appropriate position in some font that contains the requisite glyphs.

24.4.1 The list of characters

```

<!-- Special characters for HTML -->
<!-- Character entity set. Typical invocation:
<ENTITY % HTMLSpecial PUBLIC
"-//W3C//ENTITIES Special//EN//HTML">
%HTMLSpecial; -->
<!-- Portions © International Organization for Standardization 1986:
Permission to copy in any form is granted for use with
conforming SGL systems and applications as defined in
ISO 8879, provided this notice is included in all copies.
-->
<!-- Relevant ISO entity set is given unless names are newly introduced.
New names (i.e., not in ISO 8879 list) do not clash with any
existing ISO 8879 entity names. ISO 10646 character numbers
are given for each character, in hex. CDATA values are decimal
conversions of the ISO 10646 values and refer to the document
character set. Names are ISO 10646 names.
-->
<!-- C0 Controls and Basic Latin -->
<!-- ENTITY quot CDATA "&#34;" -- quotation mark = APL quote,
U+0022 ISOnum -->
<!-- ENTITY amp CDATA "&#38;" -- ampersand, U+0026 ISOnum -->
<!-- ENTITY lt CDATA "&#60;" -- less-than sign, U+003C ISOnum -->

```

```

<!ENTITY gt CDATA "&#62;" -- greater-than sign, U+003E ISONum ---
<!-- Latin Extended-A --->
<!ENTITY OE1lg CDATA "&#338;" -- latin capital ligature OE,
U+0152 ISOLat2 --->
<!ENTITY oelig CDATA "&#339;" -- latin small ligature oe, U+0153 ISOLat2 --->
<!-- ligature is a misnomer, this is a separate character in some languages --->
<!ENTITY Scaeron CDATA "&#352;" -- latin capital letter S with caron,
U+0160 ISOLat2 --->
<!ENTITY scaeron CDATA "&#353;" -- latin small letter s with caron,
U+0161 ISOLat2 --->
<!ENTITY Yuml CDATA "&#376;" -- latin capital letter Y with diaeresis,
U+0178 ISOLat2 --->

<!-- Spacing Modifier Letters --->
<!ENTITY circ CDATA "&#710;" -- modifier letter circumflex accent,
U+02C6 ISOPub --->
<!ENTITY tilde CDATA "&#732;" -- small tilde, U+02DC ISODia --->

<!-- General Punctuation --->
<!ENTITYensp CDATA "&#8194;" -- en space, U+2002 ISOPub --->
<!ENTITY emsp CDATA "&#8195;" -- em space, U+2003 ISOPub --->
<!ENTITY thinsp CDATA "&#8201;" -- thin space, U+2009 ISOPub --->
<!ENTITY zwnj CDATA "&#8204;" -- zero width non-joiner,
U+200C NEW RFC 2070 --->
<!ENTITY zwj CDATA "&#8205;" -- zero width joiner, U+200D NEW RFC 2070 --->
<!ENTITY lrm CDATA "&#8206;" -- left-to-right mark, U+200E NEW RFC 2070 --->
<!ENTITY rlm CDATA "&#8207;" -- right-to-left mark, U+200F NEW RFC 2070 --->
<!ENTITY ndash CDATA "&#8211;" -- en dash, U+2013 ISOPub --->
<!ENTITY mdash CDATA "&#8212;" -- em dash, U+2014 ISOPub --->
<!ENTITY lsquo CDATA "&#8216;" -- left single quotation mark,
U+2018 ISONum --->
<!ENTITY rsquo CDATA "&#8217;" -- right single quotation mark,
U+2019 ISONum --->
<!ENTITY sbquo CDATA "&#8218;" -- single low-9 quotation mark, U+201A NEW --->
<!ENTITY ldquo CDATA "&#8220;" -- left double quotation mark,
U+201C ISONum --->
<!ENTITY rdquo CDATA "&#8221;" -- right double quotation mark,
U+201D ISONum --->
<!ENTITY bdquo CDATA "&#8222;" -- double low-9 quotation mark, U+201E NEW --->
<!ENTITY dagger CDATA "&#8224;" -- dagger, U+2020 ISOPub --->
<!ENTITY Ddagger CDATA "&#8225;" -- double dagger, U+2021 ISOPub --->
<!ENTITY permil CDATA "&#8240;" -- per mille sign, U+2030 ISOTEch --->
<!ENTITY lsaquo CDATA "&#8249;" -- single left-pointing angle quotation mark,
U+2039 ISO proposed --->
<!-- lsaquo is proposed but not yet ISO standardized --->
<!ENTITY rsaquo CDATA "&#8250;" -- single right-pointing angle quotation mark,
U+203A ISO proposed --->
<!-- rsaquo is proposed but not yet ISO standardized --->
<!ENTITY euro CDATA "&#8364;" -- euro sign, U+20AC NEW --->

```

Appendix A: Changes

Contents

1. Changes between 24 April 1998 HTML 4.0 and 24 December 1999 HTML 4.01 versions	312
1. Changes to the specification	312
• General changes	312
• On SGML and HTML	312
• HTML Document Representation	312
• Basic HTML data types	312
• Global structure of an HTML document	313
• Language information and text direction	313
• Tables	313
• Links	313
• Objects, Images, and Applets	314
• Style Sheets in HTML Documents	314
• Frames	314
• Forms	315
• SGML Declaration	315
• Strict DTD	315
• Notes	315
• References	316
2. Errors that were corrected	316
3. Minor typographical errors that were corrected	318
4. Clarifications	322
5. Known Browser problems	322
2. Changes between 18 December 1997 and 24 April 1998 versions	322
1. Errors that were corrected	323
2. Minor typographical errors that were corrected	325
3. Changes between HTML 3.2 and HTML 4.0 (18 December 1997)	327
1. Changes to elements	327
• New elements	327
• Deprecated elements	327
• Obsolete elements	328
2. Changes to attributes	328
3. Changes for accessibility	328
4. Changes for meta data	328
5. Changes for text	328
6. Changes for links	328
7. Changes for tables	328
8. Changes for images, objects, and image maps	329
9. Changes for forms	330

10. Changes for style sheets	330
11. Changes for frames	330
12. Changes for scripting	330
13. Changes for internationalization	330

A.1 Changes between 24 April 1998 HTML 4.0 and 24 December 1999 HTML 4.01 versions

This section describes how the 24 December 1999 version of the HTML 4.01 specification differs from the 24 April 1998 version of the HTML 4.0 specification.

A.1.1 Changes to the specification

General changes

- New style sheets for the document based on W3C technical report styles.
- Added a short table of contents [p.4] .
- Updated the copyright.
- Fixed document scripts to remove markup causing crashes on some browsers.
- Thanks to Shane McCarron added to the acknowledgments [p.17] .
- In section 1.4 [p.18] , removed copyright details and refer to W3C site instead.
- References to the document character set are all ISO 10646 (and one time to UNICODE to signal equivalence). References to UNICODE refer only to the bidirectionality algorithm.
- Examples now use dated FPIs.

On SGML and HTML

- Section 3.2.2 [p.29] : Attribute values may contain colons and underscores as well.

HTML Document Representation

- The Document Character Set [p.41] : [ISO10646] now used only for references to the document character set. [UNICODE] is reserved for bidi-related references.

Basic HTML data types

- Media descriptors [p.56] : All characters in examples now described using hex notation (and reference to ISO 10646 rather than Unicode).

Global structure of an HTML document

- 7.2 HTML version information [p.60] : Add a note about the HTML WG's commitment that
 - Any changes to future HTML 4 DTDs will not invalidate documents that conform to the DTDs of the present specification. The HTML Working Group reserves the right to correct known bugs.
 - Software conforming to the DTDs of the present specification may ignore features of future HTML 4 DTDs that it does not recognize.
- 7.2 HTML version information [p.60] : Use undated, HTML 4 URIs for system identifiers. These URIs are also used globally in all examples.
- 7.4.4 Meta data [p.64] : Removed note about ongoing work at W3C on meta data and replaced with a note about RDF.
- 7.4.4.2 Meta data [p.65] : At the end of the section on HTTP headers, removed the auto-refresh example (since not part of the Recommendation) and added a note to use server-side redirects.

Language information and text direction

- The dir attribute [p.82] : Clarification that dir applies to element content, attribute values, and table direction.

Tables

- 11.2.6 Table Cells: [p.125] The definitions of rowspan and colspan changed. Now spans are bounded by groups (rowgroups or colgroups).
- 11.3.2 Table Cells: [p.132] When "char=align" not supported by the user agent, behavior is undefined.

Links

- 12.2 The A element: [p.149] The description of the type attribute for the A and (LINK) elements has been modified to emphasize its advisory nature.
- 12.2.3 Anchors with the id attribute: [p.152] It is legal for "name" and "id" to appear in the same start tag when they are both defined for an element. They must have identical values.
- 12.3.3 Links and search engines: [p.155] Removed reference to dir attribute in example since it doesn't apply to linked resources (only element content and attribute text values).
- 12.4.1 Resolving relative URIs: [p.158] Since RFC 2616 does not include a Link header field, the following statement is qualified for earlier versions of HTTP 1.1: "Link elements specified by HTTP headers are handled exactly as LINK elements that appear explicitly in a document."

Objects, Images, and Applets

- 13.2 The IMG element: [p.160] Addition of the name attribute for backwards compatibility.
- 13.2 The IMG element: [p.160] Added a note that user agents must provide different mechanisms for accessing the "longdesc" URI (of IMG) and the "src" URI (of A) when an IMG is part of the content of an A element.
- 13.3 The OBJECT element: [p.162] Added a note that when the value of "type" for OBJECT and the Content-Type HTTP header differ, the latter takes precedence.
- 13.3 The OBJECT element: [p.162] Added a statement to use PARAM instead of the "data" and "classid" attributes for OBJECT together.
- 13.4 The APPLLET element: [p.171] Added a note that, for security reasons only subdirectories are searched for the "codebase" attribute of APPLLET.
- 13.6.1 Client-side image maps: [p.174] The definition of the "poly" attribute has been cleared up. There is a note that if not closed by authors, user agents should close the polygon for the "coords" attribute of AREA.
- 13.6.1 Client-side image maps: [p.174]
 - The content model of the MAP element now allows authors to mix AREA content and block-level content.
 - User agents "should" render block-level content (used to be "may").
 - The MAP element may be used without an image for general purpose navigation tools.
 - User agents must ignore AREA elements when content is mixed (AREA and block level).
 - Authors should specify geometries completely with either AREA elements, or A elements in block content, or both.
- 13.7.2 [p.180] and 13.7.3 [p.180] : The vspace and hspace attribute definitions now look like the definitions of other attributes.
- 13.7.2 [p.180] and 13.7.3 [p.180] : The type of vspace, hspace, and border attribute values was changed from "length" to "pixels".
- 13.8 Alternate text: [p.181] The last sentence of section now links to notes for user agent developers for handling empty "alt" attribute text.

Style Sheets in HTML Documents

- 14.6 Linking to style sheets with HTTP headers: [p.194] Since RFC 2616 does not include a Link header field, the entire section is qualified to pertain only to earlier versions of HTTP 1.1.

Frames

- 16.4.1 NOFRAMES: [p.214] Added text to the NOFRAMES description about rendering when (1) frames turned off (2) frames not supported.
- 16.4.1 NOFRAMES: [p.214] Added text about which DTDs may have NOFRAMES (frames, transitional).

Forms

- 17.2.1 Control types: [p.221] In the description of radio buttons, when no radio button is initially selected, user agent behavior for selecting one is undefined. This differs from RFC 1866.
- 17.3 The FORM element: [p.222] Addition of the `name` attribute for backwards compatibility.
- 17.3 The FORM element: [p.222] Removed the reference to the "mailto" URI in the "action" attribute definition.
- 17.3 The FORM element: [p.222] Removed "mailto" example near end of section since behavior not defined in this spec.
- 17.3 The FORM element: [p.222] The `accept` attribute is added to the DTD fragment. Also, the description of the `accept_charset` attribute is amended.
- 17.4 The INPUT element: [p.224] Added missing "ismap" for the INPUT element. Also, in definition of `value`, add "checkbox" to values of `type` that require a value.
- 17.6.1 [p.231] : When no option is preselected, user agent behavior is undefined. Authors should supply and explicit none option to cover this case. This behavior differs from RFC 1866.

SGML Declaration

- SGML Declaration of HTML 4: [p.263] Removed text about up-to-date references to ISO 10646. Replaced with: "Revisions of the HTML 4 specification may update the reference to ISO 10646 to include additional changes."

Strict DTD

- `vspace/hspace` attributes for `IMG`, `OBJECT`, `APPLET` in pixels.
- Changed content model of `MAP` to `((%block); | AREA)+`
- Added "ismap" attribute to `INPUT`
- The `accept` attribute is added to the DTD fragment for the `FORM` element.
- The `axis` attribute comment has been changed to refer to a comma-separated list.
- The `archive` attribute for the `OBJECT` element takes a value of type `CDATA` instead of type `%URI` since the value is a space-separated list of URIs.

Notes

- Notes [p.350] Updated notes on accessibility to point to Web Content Accessibility Guidelines.

References

- Updated links to RFCs to use `http://www.ietf.org/rfc`
- Put links in titles.
- Added revised date of 27 Aug 1998 for [DATETIME]
- Added revised date of 11 Jan 1999 for [CSS1].
- Publication date of [CSS2] fixed.
- [UNICODE] has been updated to version 3.0
- [ISO10646] has been updated to allow for new character assignments. Note that amendment five is specifically taken into account.
- [RFC1766] expected to be updated.
- [RFC2279] obsoletes [RFC2044].
- [RFC2616] obsoletes [RFC2068].
- [RFC2388] added in addition to [RFC1867].
- [LEXHTML] address updated, date added.
- [DCORE] address updated.
- Updated [WEBSGML]
- [HTML3STYLE] address updated.
- Added [RDF10] (replaced old RDF)
- Changed [WAIGUIDE] -> [WAI]
- Added informative references [WCGL], [UAAG], and [ATGL]
- Updated URI reference to [URI] (RFC 2396).

A.1.2 Errors that were corrected

Section 13.6.1 [p.174]

Image map examples using "poly" have been fixed to form a closed polygon. Also, the last pair of coordinates is the same as the first to close the polygon.

Section 14.4.1 [p.192]

In the final example, the `STYLE` element is missing the attribute assignment "media=screen, print".

Section 15.2.1 [p.199]

In the example with "mypar", the CSS rule should read

```
P#mypar {font-style: italic; color: blue}
```

In CSS, "#" refers to an ID name while the "." refers to a class name. This example is dealing with the "id" attribute.

Section 16.2.2 [p.209]

Values for `marginwidth` and `marginheight` must be 0 pixels or more, not 1 pixel or more.

Section 16.2.2 [p.209]

The `FRAME` element does not take the `target` attribute.

Section 16.5 [p.217]

The `IFRAME` element does not take the `target` attribute.

Section 17.2.1 [p.221]

In the description of "checkboxes", change "selected" to "checked" in "when the control element's selected attribute is set."

Section 17.6.1 [p.231]

In the "Attributes defined elsewhere" section for the `OPTGROUP` element, the attributes `onfocus`, `onblur`, and `onchange` should not be there.

Section 18.2.3 [p.254]

To the list of elements that take `onfocus` and `onblur`, add `A` and `AREA`.

Section 20 [p.263]

The SGML Declaration for HTML 4 must be modified slightly to support hexadecimal numeric character references. The lines:

```
DELIM
GENERAL SGMREF
SHORTREF SGMREF
```

must be changed to:

```
DELIM
GENERAL SGMREF
HCRO "&#38;#x" -- 38 is the number for ampersand --
SHORTREF SGMREF
```

And the initial `<!SGML "ISO 8879:1986" must be changed to <!SGML`

```
"ISO 8879:1986 (WWW)".
```

Section 21 [p.265]

The `HR` element should also take the `lang` and `dir` attributes. These are noted as being defined elsewhere at the element's definition, but were left out of the DTDs.

Section 21 [p.265]

The `OBJECT` element's archive attribute is defined in the DTD as taking a value of type `%URI`". This is incorrect: the value may be a space-separated list of URIs (as indicated in the definition of the attribute and in the DTD comment). The `FORM` element's DTD fragment should include a definition for the `accept` attribute, which is listed in the element's definition. The definition should be the following:

```
accept %ContentType; #IMPLIED -- list of MIME types for file upload --
```

Section B.4.1 [p.340]

At the end of the section, the following sentences are incorrect: "The list of terms in the content is ALL, INDEX, NOFOLLOW, NOINDEX. The name and the content attribute values are case-insensitive." In fact, the `META` definition specifies that values for the `name` and `content` attributes are case-sensitive.

Section B.4.1.1 [p.340]

The specification reads, "Blank lines are not permitted." Blank lines are permitted in the robots.txt file, just not within a single "record". Note that the

specification doesn't define record.

Further down the page, the specification reads, "There must be exactly one "User-agent" field per record." In fact, there can be more than one User-Agent field in the robots.txt file, just not more than one per record.

For information about search robots, please consult, for example:

- <http://www.kollar.com/robots.html>
- <http://info.webcrawler.com/mak/projects/robots/norobots-rfc.html>
- <http://info.webcrawler.com/mak/projects/robots/robots.html>

References [p.353]

- The [URI] [p.355] reference should be updated to RFC 2396 as of August 1998. "Uniform Resource Identifiers (URI): Generic Syntax", T. Berners-Lee, R. Fielding, L. Masinter, August 1998. RFC 2396 updates [RFC1738] and [RFC1808].

A.1.3 Minor typographical errors that were corrected

Section 2.1.1 [p.19]

The phrase "accessible via the path "/TR/REC-html4"/". should end with "/TR/REC-html40/".

Section 2.1.3 [p.20]

In the third bullet, the word "applets" should be "applet".

Section 3.3 [p.28]

In bullet two, the sentence "Whether the element's end tag is optional." should read "Whether the element's tags are optional."

Section 3.2.1 [p.28]

In the sentence beginning "Please consult the SGML standard", the phrase "an end tag closes all omitted start tags up to the matching start tag (section 7.5.1)" should read "an end tag closes, back to the matching start tag, all unclosed intervening start tags with omitted end tags".

Section 3.2.2 [p.29]

"Attribute names are always case-insensitive" is missing a final period.

Section 3.3.4.2 [p.36]

The example with the `OPTION` element has an improper end tag; it should be `</OPTION>`.

Later in the section, the sentence that begins "Authors should be aware than" should say "aware that" instead.

Section 5.2.2 [p.43]

Change "ASCII characters" to "ASCII-valued bytes".

Section 5.3.1 [p.45]

The second bullet should read "a" instead of "an" in "where H is an hexadecimal number".

Section 6.5.1 [p.52]

The first sentence needs the indefinite article "a" before the word "document".

- Section 6.10 [p.53]
The first sentence needs the indefinite article "a" before the word "single".
- Section 6.12 [p.54]
Under "Next", "in an linear" should read "in a linear" instead.
- Section 6.16 [p.57]
Change "cancelling" to "canceling".
- Section 7.4.4.3 [p.68]
In the paragraph beginning "The scheme attribute allows...", replace "Month-Date-Year" with "Month-Day-Year".
- Section 7.5.4 [p.73]
In the sentence after the example, make "declaration" plural.
- Section 7.5.6 [p.76]
For the ADDRESS element, in the section "Attributes defined elsewhere", style and title are missing.
Also, after the section on "Attributes defined elsewhere", in "contact information for document", put "a" before "document".
- Section 8.2.3 [p.84]
In "Authors may also use special Unicode characters to achieve multiply" change to "multiple" at the end.
- Section 11.2.4.1 [p.118]
The sentence "The first COL element refers to the first 39 columns (doing nothing special to them) and the second one assigns an id value to the fortieth columns so that style sheets may refer to it." should have "fortieth column" instead.
- Section 11.2.5 [p.124]
For the TR element, in the section "Attributes defined elsewhere", bgColor is missing.
- Section 11.2.6 [p.125]
For the TH and TD elements, the type of the width and height attributes is changed from "%Pixels;" to "%Length;" to allow for percentage values.
- Section 11.3.1 [p.130]
In the first sentence of the frame attribute definition, use "surrounding" instead of "that surrounds".
- Section 11.4.1 [p.136]
First bullet, third sentence. "Note that its not always possible" should have "it's" instead.
- Section 12.1.2 [p.147]
The last sentence should read "Further information is given below on using links for..." (change "of" to "on"). This sentence is also missing its closing punctuation.
- Section 12.2.2 [p.152]
The last paragraph should read "Since the DTD defines the LINK element to be empty..." (insert definite article "the" before "LINK").
- Section 12.2.3 [p.152]
Just before section 12.2.4, the third bullet. "richer anchors names" should read "richer anchor names".

- Section 13.3.4 [p.169]
In the paragraph that begins "In the following example...", the phrase "cause it so be instantiated" should be changed to cause it to be instantiated" (change "so" to "to").
- Section 13.4 [p.171]
Just after the deprecated example, the sentence "This example may be rewritten as follows with OBJECT as follows." should say "This example may be rewritten with OBJECT as follows:".
- Section 13.6.1 [p.174]
Under the "coords" attribute, the word "and" should be substituted for the word "a" so the sentence reads, "This attribute specifies the position and shape on the screen."
- Section 13.7.1 [p.179]
In the definition of the height attribute, the phrase "Image and object override" should read "Image and object height override".
- Section 15.1.3.1 [p.197]
Under the subheading "Float an object", in the first paragraph, the first use of the word "object" should be "objects".
- Section 15.1.3.2 [p.198]
In the "Deprecated" example, the first sentence should read "If the clear attribute is set to left or all, the next line will appear as follows:" ("the" before "next line").
- Section 15.3 [p.202]
The align attribute for HR is not defined elsewhere.
- Section 16.1 [p.205]
In the last sentence of the first paragraph, the word "though" should be "through".
- Section 16.3.1 [p.213]
In the second sentence, the word "factorizing" should be "factoring".
- Section 16.4.1 [p.214]
The list of "attributes defined elsewhere" was inadvertently omitted after the definition of NOFRAMES. These attributes are: class, id, lang, dir, title, style, and the %events; [p.282] attributes.
- Section 17.1 [p.219]
In "(entering text, selecting menu items, etc.)", add the ", " after "text".
- Section 17.5 [p.228]
In the paragraph that begins "Visual user agents may render...", the indefinite article "a" should be removed from before the word "flat".
- Section 17.12.1 [p.244]
A comma should be added between BUTTON and INPUT in the list of elements that support the "disabled" attribute.
- Section 17.13.4.2 [p.248]
In the examples at the end of the section, change "Content-Disposition: attachment" to "Content-Disposition: file". Also, in an earlier example, change "server.dom" to "server.com".

Section 18.2.2.1 [p.253]

After the first example, the indefinite article before "content-type" needs to be "a", not "an". The same applies to "content-type" in the next paragraph.

In the sentence beginning "Documents that do not specify...", the indefinite article "a" needs to be removed from before "default scripting language information".

Section 18.2.3 [p.254]

In the first sentence of the first note, the word "realm" should be preceded by the definite article "the".

Section 18.3.1 [p.258]

In the second sentence of the first paragraph, the word "be" needs to be inserted between the words "only" and "rendered".

Section 21 [p.265]

- In all DTDS, under the COLGROUP element, the content model should indicate "COL", not "col".
- In the comment about the %Scope entity, change "axes" to "headers" attribute.

Section 24.2.1 [p.300]

At end of definition of "thorn", remove stray final word.

Section 24.4 [p.308]

Change "cp-1252" to "CP-1252".

Appendix: Changes for tables [p.328]

In the paragraph on the COLGROUP element, the last sentence should read: "The semantics of COLGROUP have been clarified over previous drafts, and rules="basic" has been replaced by rules="groups"."

Changes to elements [p.327]

The list of deprecated elements should include S.

Section B.3.2 [p.336]

In "delimiter followed by a name character", change to delimiter followed by a name start character".

Section B.4 [p.339]

Under "Provide keywords and descriptions", the middle of the sentence "The value of the name attribute sought by a search attribute is not defined by this specification." should read "search engine" instead.

Section B.4 [p.339]

In the example to indicate the beginning of a collection replace `rel="begin"` with `rel="start"`.

Section B.4.1 [p.340]

Remove "The name and the content attribute values are case-insensitive."

Section B.5.1.2 [p.342]

The last sentence of the last paragraph is missing a closing parenthesis.

Section B.7.1.1 [p.348]

In the deprecated example:

```
<BODY bgcolor=&{randomrgb} ; >
```

The word "randomrgb" should be "randomrgb".

A.1.4 Clarifications

Section 3.2.1 [p.28]

In seventh paragraph, added "back to the matching start tag" to "(e.g., they must be properly nested, an end tag closes, back to the matching start tag, all unclosed intervening start tags with omitted end tags (section 7.5.1), etc.)."

Section 3.2.4 [p.30]

Added a statement that comments are markup.

Section 3.3.3 [p.32]

In the second list item, change "Whether the element's end tag" to "Whether the element's tags".

Section 3.3.3.1 [p.33]

In a content model definition, "A" means that "A" must occur one time and only one time. Also, added "+(A)" and "-(A)" to the section on content model syntax.

Section 7.4.2 [p.62]

Clarified that TITLE may not include comments.

Section 10.3 [p.106]

All uses of "cracker" in this section and its subsections are replaced with "hacker". Also, definitions of "hacker" and "nerd" taken from "The Hacker's Dictionary".

Section 13.7.2 [p.180]

The `hspace` and `vspace` attributes are deprecated.

Section 13.7.4 [p.180]

The `align` attribute is deprecated for `IMG`, `OBJECT`, and `APPLET`.

A.1.5 Known Browser problems

- Some versions of Netscape Navigator 4.0X crash upon reading Chapter 3 of previous versions of this specification. Netscape is aware of this bug and have fixed it in version 4.5. To work around this bug, go to the Edit/Preferences/Advanced submenu and disable Style Sheets (and possibly JavaScript).

A.2 Changes between 18 December 1997 and 24 April 1998 versions

This section describes how the 24 April 1998 version of the HTML 4.0 specification differs from the 18 December 1997 version.

A.2.1 Errors that were corrected

Section 2.1.1 [p.19]

"http://www.w3.org/TR/PR-html4/cover.html" was said to designate the current HTML specification. The current HTML specification is actually at <http://www.w3.org/TR/REC-html40>.

Section 7.5.2 [p.71]

The hypertext link on `name` was incorrect. It now links to `types.html#type-name` [p.50].

Section 7.5.4 [p.73]

`href` was listed as an attribute of the `DIV` and `SPAN` elements. It is not.

Section 7.5.6 [p.76]

A `P` element was used in the example. It is invalid in `ADDRESS`.

Section 8.1 [p.79]

In the first example, which reads "Her super-powers were the result...", there was an extra double quote mark before the word "Her".

Section 9.3.4 [p.97]

The attribute `width` [p.97] was not noted as **deprecated** [p.38].

Section 11.2.4, "Calculating the width of columns" [p.122]

The sentence "We have set the value of the align attribute in the third column group to 'center'" read "second" instead of "third".

Section 11.2.6, "Cells that span several rows or columns" [p.128]

The second paragraph read "In this table definition, we specify that the cell in row four, column two should span a total of three columns, including the current row." It now ends "...including the current column."

Section 13.2 [p.160]

The sentence beginning "User agents must render alternate text when they cannot support ..." read "next", instead of "text".

Section 13.6.2 [p.179]

The last sentence of the second paragraph applied to both the `IMG` and `INPUT` elements. However, the `ismap` attribute is not defined for `INPUT`. The sentence now only applies to `IMG`.

Section 14.2.3 [p.187]

The `title` attribute for the `STYLE` element was not listed as an attribute defined elsewhere.

Section 14.3.2 [p.191]

The second example set `title="Compact"`. It now sets `title="compact"`.

Section 15.1.2 [p.195]

The sentence ending "the align attribute." read "the align element."

Section 15.1.3.2 [p.198]

The CSS style rule "BR.mybr { clear: left }" was incorrect, since it refers to the class "mybr" and not the id value. The correct syntax is: "BR#mybr { clear: left }".

Section 16 [p.205]

All the examples containing a Document Type Declaration used something like "THE_LATEST_VERSION_/frameset.dtd" or "THE_LATEST_VERSION_" as the system identifier for the Frameset DTD. They now use the proper document

type declaration indicated in Section 7.2 [p.60]

Section 16.3 [p.212] and Section 16.3.1 [p.213]

The second example of 16.3 and the example of 16.3.1 used the wrong DTD; they now use the Transitional DTD.

Section 17.5 [p.228]

In "attributes defined elsewhere" for the `BUTTON` element, `id`, `class`, `lang`, `dir`, `title`, `style`, and `tabindex` were missing. Also, `usemap` has been removed.

Section 17.6/17.6.1 [p.230]

The "attributes defined elsewhere" for `OPTION` and `OPTGROUP` mistakenly listed `onfocus`, `onblur`, and `onchange`. The "attributes defined elsewhere" section was missing for the `SELECT` element (please see the DTD for the full list of attributes).

Section 17.9.1 [p.237]

The `tabindex` attribute was said to be defined for the `LABEL` element. It is not.

Section 17.12.2 [p.244]

The sentence "The following elements support the `readonly` attribute: `INPUT` and `TEXTAREA`." read "The following elements support the `readonly` attribute: `INPUT`, `TEXT`, `PASSWORD`, and `TEXTAREA`."

Section 18.2.2, "Local declaration of a scripting language" [p.254]

The first paragraph read: "It is also possible to specify the scripting language in each `SCRIPT` element via the `type` attribute. In the absence of a default scripting language specification, this attribute must be set on each `SCRIPT` element." Since the `type` attribute is required for the `SCRIPT` element, this paragraph now reads: "The `type` attribute must be specified for each `SCRIPT` element instance in a document. The value of the `type` attribute for a `SCRIPT` element overrides the default scripting language for that element."

Section 21 [p.265]

Added note that the spec includes some syntactic constraints that cannot be expressed in the DTD.

Section 24.2.1 [p.300] and file `HTMLat1.ent`

The comment for the character reference "not" read "=" discretionary hyphen". This has been removed.

The FPI in comment read `"-//W3C/ENTITIES Full Latin 1//EN/HTML"`, instead this is now `"-//W3C/ENTITIES Latin1//EN/HTML"`.

Section 24.3.1 [p.304] and file `HTMLsymbol.ent`

The FPI in comment read `"-//W3C/ENTITIES Symbolic//EN/HTML"`, instead this is now `"-//W3C/ENTITIES Symbols//EN/HTML"`.

Section A.1.1, "New elements" [p.312] (previously A.1.1) and Section A.1.1, "Deprecated elements" [p.312] (previously A.1.2)

The `S` element which is **deprecated** [p.38] was listed as part of the changes between HTML 3.2 and HTML 4.0. This element was not actually defined in HTML 3.2 [p.356]. It is now in the new elements list.

Section A.1.3 (previously A.3) [p.312]

The `longdesc` attribute was said to be specified for tables. It is not. Instead, the `summary` attribute allows authors to give longer descriptions of tables.

Section B.4 [p.339]

The sentence "You may help search engines by using the `LINK` element with `rel="start"` along with the `title` attribute, ..." read "You may help search engines by using the `LINK` element with `rel="begin"` along with a `TITLE`, ..." The same stands for the companion example.

Section B.5.1 [p.342]

The sentence "This can be altered by setting the `width` attribute of the `TABLE` element." read "This can be altered by setting the `width-TABLE` attribute of the `TABLE` element."

Section B.5.2 [p.344]

The sentence "Rules for handling objects too large for a column apply when the explicit or implied alignment results in a situation where the data exceeds the assigned width of the column." read "too large for column". The meaning of the sentence was unclear since it referred to "rules" governing an error condition; user agent behavior in error conditions lies outside the scope of the specification.

Index of attributes [p.363]

The `href` attribute for the `BASE` element was marked as **deprecated** [p.38] . It is not. However, it is not defined in the Strict DTD either.

The `language` attribute for the `SCRIPT` element was not marked as **deprecated** [p.38] . It is now, and it is no longer defined in the Strict DTD.

A.2.2 Minor typographical errors that were corrected

Section 2.1.3 [p.20]

"Relative URIs are resolved ..." was "Relative URIs are resolved ...".

Section 2.2.1 [p.21]

The second word "of" was missing in "Despite never receiving consensus in standards discussions, these drafts led to the adoption of a range of new features."

Section 3.3.3 [p.32]

The sentence "Element types that are designed to have no content are called empty elements." contained one too many "elements". The word "a" was missing in the sentence "A few HTML element types use an additional SGML feature to exclude elements from a content model".

Also, in list item two, a period was missing between "optional" and "Two".

Section 3.3.4 [p.34]

In the section on "Boolean attributes", the sentence that begins "In HTML, boolean attributes may appear in minimized ..." included a bogus word "be".

Section 6.3 [p.50]

The sentence beginning "For introductory information about attributes, ..." read "For introductory about attributes, ...".

Section 6.6 [p.52]

In the first sentence of the section on Pixels, "is an integer" read "is integer".

Section 7.4.1 [p.62]

The first word "The" was missing at the beginning of the section title.

Section 7.4.4 [p.64]

The last word "a" was missing in the sentence "The meaning of a property and the set of legal values for that property should be defined in a reference lexicon called profile."

Section 7.5.2 [p.71]

"Variable déclarée deux fois" read "Variable déclarée deux fois".

Section 9.2.2 [p.92]

The language of the quotations was "en" instead of "en-us", while in British English, the single quotation marks would delimit the outer quotation.

Section 9.3.2 [p.95]

In the first line, the sixth character of "
" was the letter 'O' instead of a zero.

Section 10.3.1 [p.108]

"(they are case-sensitive)" read "(the are case-sensitive)".

Section 12.1.1 [p.145]

In the sentence beginning "Note that the `href` attribute in each source ..." the space was missing between "href" and "attribute".

Section 12.1.2 [p.147]

The sentence "Links that express other types of relationships have one or more link types specified in their source anchors." read "Links that express other types of relationships have one or more link type specified in their source anchor."

Section 12.1.5 [p.148]

The second paragraph reads "the `hreflang` attribute provides user agents about the language of a ..." It should read "the `hreflang` attribute provides user agents with information about the language of a ..."

Section 13.3.2 [p.167]

In the sentence beginning "Any number of `PARAM` elements may appear in the content of an `OBJECT` or `APPLET` element, ..." a space was missing between "APPLET" and "element".

Section 14.2.2 [p.186]

There was a bogus word "style" at the beginning of the sentence "The `style` attribute specifies ..."

Section 17.2 [p.220]

In "Those controls for which name/value pairs are submitted are called successful controls" the word "for" was missing.

Section 17.10 [p.239]

There was a bogus word "/samp" just before section 17.11.

Section 17.11 [p.241]

The first sentence read, "In an HTML document, an element must receive focus from the user in order to become active and perform their tasks" (instead of "its tasks").

Section 18.2.2 [p.253]

Just before section 18.2.3, the sentence that includes "a name attribute takes

precedence over an id if both are set." read "over a id if both are set."
 Section 19.1 [p.261]
 The section title read "document Document Validation": It now is "Document Validation".

Section 21 [p.265]

The FPI for the Transitional HTML 4.0 DTD was missing a closing double quote.
 Section B.5.1/B.5.2 [p.342]

This sections referred to a non-existent `cols` attribute. This attribute is not part of HTML 4.0. Calculating the number of columns in a table is described in section Section 11.2.4.3 [p.121], in the chapter on tables. In sections B.5.1 and B.5.2, occurrences of `cols` have been replaced by "the number of columns specified by the `col` and `colgroup` elements".

Section B.5.2 [p.344]

In the sentence "The values for the frame attribute have been chosen to avoid clashes with the rules, align and valign attributes." a space was missing between "the" and "frame" and the last attribute was "valign-COLGROUP".

Section B.10.1 [p.350]

The last sentence read "Once a file is uploaded, the processing agent should process and store the it appropriately." "the it" was changed to "it".

Index of Elements [p.359]

"strike-through" in the description of the `s` element read "sstrike-through".

A.3 Changes between HTML 3.2 and HTML 4.0 (18 December 1997)

This section describes how the 18 December 1997 specification of HTML 4.0 differs from HTML 3.2 (HTML32) [p.356]).

A.3.1 Changes to elements

New elements

The new elements in HTML 4.0 are: `ABBR`, `ACRONYM`, `BDO`, `BUTTON`, `COL`, `COLGROUP`, `DEL`, `FIELDSET`, `FRAME`, `FRAMESET`, `IFRAME`, `INS`, `LABEL`, `LEGEND`, `NOPRAMES`, `NOSCRIPT`, `OBJECT`, `OPTGROUP`, `PARAM`, `S` (deprecated), `SPAN`, `TBODY`, `TFOOT`, `THEAD`, and `Q`.

Deprecated elements

The following elements are deprecated [p.38]: `APPLET`, `BASEFONT`, `CENTER`, `DIR`, `FONT`, `ISINDEX`, `MENU`, `S`, `STRIKE`, and `U`.

Obsolete elements

The following elements are obsolete: `LISTING`, `PLAINTEXT`, and `XMP`. For all of them, authors should use the `PRE` element instead.

A.3.2 Changes to attributes

- Almost all attributes that specify the presentation of an HTML document (e.g., colors, alignment, fonts, graphics, etc.) have been deprecated [p.38] in favor of style sheets. The list of attributes [p.363] in the appendix indicates which attributes have been deprecated [p.38].
- The `id` and `class` attribute allow authors to assign name and class information [p.71] to elements for style sheets, as anchors, for scripting, for object declarations, general purpose document processing, etc.

A.3.3 Changes for accessibility

HTML 4.0 features many changes to promote accessibility [p.22], including:

- The `title` attribute may now be set on virtually every element.
- Authors may provide long descriptions of tables (see the `summary` attribute), images and frames (see the `longdesc` attribute).

A.3.4 Changes for meta data

Authors may now specify profiles [p.68] that provide explanations about meta data specified with the `META` or `LINK` elements.

A.3.5 Changes for text

- New features for internationalization [p.330] allow authors to specify text direction and language.
- The `INS` and `DEL` elements allow authors to mark up changes in their documents.
- The `ABBR` and `ACRONYM` elements allow authors to mark up abbreviations and acronyms in their documents.

A.3.6 Changes for links

- The `id` attribute makes any element the destination anchor of a link.

A.3.7 Changes for tables

The HTML 4.0 table model has grown out of early work on HTML+ and the initial draft of HTML3.0 [p.355]. The earlier model has been extended in response to requests from information providers as follows:

- Authors may specify tables that may be incrementally displayed as the user agent receives data.
- Authors may specify tables that are more accessible to users with non-visual user agents.
- Authors may specify tables with fixed headers and footers. User agents may take advantage of these when scrolling large tables or rendering tables to paged media.

The HTML 4.0 table model also satisfies requests for optional column-based defaults for alignment properties, more flexibility in specifying table frames and rules, and the ability to align on designated characters. It is expected, however, that style sheets [p.183] will take over the task of rendering tables in the near future.

In addition, a major goal has been to provide backwards compatibility with the widely deployed Netscape implementation of tables. Another goal has been to simplify importing tables conforming to the SGML CALS model. The latest draft makes the `align` attribute compatible with the latest versions of the most popular browsers. Some clarifications have been made to the role of the `dir` attribute and recommended behavior when absolute and relative column widths are mixed.

A new element, `COLGROUP`, has been introduced to allow sets of columns to be grouped with different width and alignment properties specified by one or more `COL` elements. The semantics of `COLGROUP` have been clarified over previous drafts, and `rules="basic"` has been replaced by `rules="groups"`.

The `style` attribute is included as a means for extending the properties associated with edges and interiors of groups of cells. For instance, the line style: dotted, double, thin/thick etc; the color/pattern fill for the interior, cell margins and font information. This will be the subject for a companion specification on style sheets.

The `frame` and `rules` attributes have been modified to avoid SGML name clashes with each other, and to avoid clashes with the `align` and `valign` attributes. These changes were additionally motivated by the desire to avoid future problems if this specification is extended to allow `iframe` and `rules` attributes with other table elements.

A.3.8 Changes for images, objects, and image maps

- The `OBJECT` element allows generic inclusion of objects.
- The `IFRAME` and `OBJECT` elements allow authors to create embedded documents.
- The `alt` attribute is required on the `IMG` and `AREA` elements.
- The mechanism for creating image maps [p.173] now allows authors to create more accessible image maps. The content model of the `MAP` element has changed for this reason.

A.3.9 Changes for forms

This specification introduces several new attributes and elements that affect forms:

- The `accesskey` attribute allows authors to specify direct keyboard access to form controls.
- The `disabled` attribute allows authors to make a form control initially insensitive.
- The `readonly` attribute, allows authors to prohibit changes to a form control.
- The `LABEL` element associates a label with a particular form control.
- The `FIELDSET` element groups related fields together and, in association with the `LEGEND` element, can be used to name the group. Both of these new elements allow better rendering and better interactivity. Speech-based browsers can better describe the form and graphic browsers can make labels sensitive.
- A new set of attributes, in combination with scripts [p.251], allow form providers to verify user-entered data.
- The `BUTTON` element and `INPUT` with `type` set to "button" can be used in combination with scripts [p.251] to create richer forms.
- The `OPTGROUP` element allows authors to group menu options together in a `SELECT`, which is particularly important for form accessibility.
- Additional changes for internationalization [p.330].

A.3.10 Changes for style sheets

HTML 4.0 supports a larger set of media descriptors [p.56] so that authors may write device-sensitive style sheets.

A.3.11 Changes for frames

HTML 4.0 supports frame documents and inline frames.

A.3.12 Changes for scripting

Many elements now feature event attributes [p.254] that may be coupled with scripts; the script is executed when the event occurs (e.g., when a document is loaded, when the mouse is clicked, etc.).

A.3.13 Changes for internationalization

HTML 4.0 integrates the recommendations of [RFC2070] [p.356] for the internationalization of HTML.

However, this specification and [RFC2070] [p.356] differ as follows:

- The `accept-charset` attribute has been specified for the `FORM` element rather than the `TEXTAREA` and `INPUT` elements.
- The HTML 4.0 specification makes additional clarifications with respect to the

- bidirectional algorithm [p.82] .
 - The use of CDATA [p.50] to define the `SCRIPT` and `STYLE` elements does not preserve the ability to transcode documents, as described in section 2.1 of [RFC2070] [p.356] .

Appendix B: Performance, Implementation, and Design Notes

Contents

1. Notes on invalid documents	334
2. Special characters in URI attribute values	334
1. Non-ASCII characters in URI attribute values	334
2. Ampersands in URI attribute values	335
3. SGML implementation notes	335
1. Line breaks	335
2. Specifying non-HTML data	336
• Element content	336
• Attribute values	337
3. SGML features with limited support	337
4. Boolean attributes	337
5. Marked Sections	338
6. Processing Instructions	338
7. Shorthand markup	338
4. Notes on helping search engines index your Web site	339
1. Search robots	340
• The robots.txt file	340
• Robots and the META element	341
2. Design rationale	342
• Dynamic reformatting	342
• Incremental display	342
• Structure and presentation	343
• Row and column groups	344
• Accessibility	344
2. Recommended Layout Algorithms	344
• Fixed Layout Algorithm	345
• Autolayout Algorithm	345
6. Notes on forms	347
1. Incremental display	348
2. Future projects	348
7. Notes on scripting	348
1. Reserved syntax for future script macros	348
• Current Practice for Script Macros	348
8. Notes on frames	350
9. Notes on accessibility	350
10. Notes on security	350
1. Security issues for forms	350

The following notes are informative, not normative. Despite the appearance of words such as "must" and "should", all requirements in this section appear elsewhere in the specification.

B.1 Notes on invalid documents

This specification does not define how conforming user agents handle general error conditions, including how user agents behave when they encounter elements, attributes, attribute values, or entities not specified in this document.

However, to facilitate experimentation and interoperability between implementations of various versions of HTML, we recommend the following behavior:

- If a user agent encounters an element it does not recognize, it should try to render the element's content.
- If a user agent encounters an attribute it does not recognize, it should ignore the entire attribute specification (i.e., the attribute and its value).
- If a user agent encounters an attribute value it doesn't recognize, it should use the default attribute value.
- If it encounters an undeclared entity, the entity should be treated as character data.

We also recommend that user agents provide support for notifying the user of such errors.

Since user agents may vary in how they handle error conditions, authors and users must not rely on specific error recovery behavior.

The HTML 2.0 specification (RFC1866 [p.356]) observes that many HTML 2.0 user agents assume that a document that does not begin with a document type declaration refers to the HTML 2.0 specification. As experience shows that this is a poor assumption, the current specification does not recommend this behavior.

For reasons of interoperability, authors must not "extend" HTML through the available SGML mechanisms (e.g., extending the DTD, adding a new set of entity definitions, etc.).

B.2 Special characters in URI attribute values

B.2.1 Non-ASCII characters in URI attribute values

Although URIs do not contain non-ASCII values (see [URI] [p.355], section 2.1) authors sometimes specify them in attribute values expecting URIs (i.e., defined with %URI; [p.266] in the DTD [p.265]). For instance, the following href value is **illegal**:

```
<A href="http://foo.org/Häkön">...</A>
```

We recommend that user agents adopt the following convention for handling non-ASCII characters in such cases:

1. Represent each character in UTF-8 (see [RFC2279] [p.354]) as one or more bytes.
2. Escape these bytes with the URI escaping mechanism (i.e., by converting each byte to %HH, where HH is the hexadecimal notation of the byte value).

This procedure results in a syntactically legal URI (as defined in [RFC1738] [p.354] , section 2.2 or [RFC2141] [p.354] , section 2) that is independent of the character encoding [p.41] to which the HTML document carrying the URI may have been transcoded.

Note. *Some older user agents trivially process URIs in HTML using the bytes of the character encoding [p.41] in which the document was received. Some older HTML documents rely on this practice and break when transcoded. User agents that want to handle these older documents should, on receiving a URI containing characters outside the legal set, first use the conversion based on UTF-8. Only if the resulting URI does not resolve should they try constructing a URI based on the bytes of the character encoding [p.41] in which the document was received.*

Note. *The same conversion based on UTF-8 should be applied to values of the name attribute for the A element.*

B.2.2 Ampersands in URI attribute values

The URI that is constructed when a form is submitted [p.245] may be used as an anchor-style link (e.g., the href attribute for the A element). Unfortunately, the use of the "&" character to separate form fields interacts with its use in SGML attribute values to delimit character entity references [p.30] . For example, to use the URI "http://host/?x=1&y=2" as a linking URI, it must be written or .

We recommend that HTTP server implementors, and in particular, CGI implementors support the use of "." in place of "&" to save authors the trouble of escaping "&" characters in this manner.

B.3 SGML implementation notes

B.3.1 Line breaks

SGML (see [ISO8879] [p.353] , section 7.6.1) specifies that a line break immediately following a start tag must be ignored, as must a line break immediately before an end tag. This applies to all HTML elements without exception.

The following two HTML examples must be rendered identically:

```
<P>Thomas is watching TV.</P>
<P>
Thomas is watching TV.
</P>
```

So must the following two examples:

```
<A>My favorite Website</A>
<A>
My favorite Website
</A>
```

B.3.2 Specifying non-HTML data

Script [p.251] and style [p.183] data may appear as element content or attribute values. The following sections describe the boundary between HTML markup and foreign data.

Note. *The DTD [p.265] defines script and style data to be CDATA for both element content and attribute values. SGML rules do not allow character references [p.45] in CDATA element content but do allow them in CDATA attribute values. Authors should pay particular attention when cutting and pasting script and style data between element content and attribute values.*

This asymmetry also means that when transcoding from a richer to a poorer character encoding, the transcoder cannot simply replace unconvertible characters in script or style data with the corresponding numeric character references; it must parse the HTML document and know about each script and style language's syntax in order to process the data correctly.

Element content

When script or style data is the content of an element (SCRIPT and STYLE), the data begins immediately after the element start tag and ends at the first ETAGO ("</") delimiter followed by a name start character ([a-zA-Z]); note that this may not be the element's end tag. Authors should therefore escape "</" within the content. Escape mechanisms are specific to each scripting or style sheet language.

ILLEGAL EXAMPLE:

The following script data incorrectly contains a "</" sequence (as part of "") before the SCRIPT end tag:

```
<SCRIPT type="text/javascript">
document.write ("<EM>This won't work</EM>")
</SCRIPT>
```

In JavaScript, this code can be expressed legally by hiding the ETAGO delimiter before an SGML name start character:


```
<SCRIPT type="text/javascript">
document.write ("<EM>This will work</EM>")
</SCRIPT>
```

In Tcl, one may accomplish this as follows:

```
<SCRIPT type="text/tcl">
document write "<EM>This will work</EM>"
</SCRIPT>
```

In VBScript, the problem may be avoided with the Chr() function:

```
"<EM>This will work<" & Chr(47) & "EM>"
```

Attribute values

When script or style data is the value of an attribute (either style or the intrinsic event [p.254] attributes), authors should escape occurrences of the delimiting single or double quotation mark within the value according to the script or style language convention. Authors should also escape occurrences of "&" if the "&" is not meant to be the beginning of a character reference [p.45].

- '"' should be written as """ or """
- '&' should be written as "&" or "&"

Thus, for example, one could write:

```
<INPUT name="num" value="0"
onchange="if (compare(this.value, &quot;help&quot;)) {gethelp()} ">
```

B.3.3 SGML features with limited support

SGML systems conforming to [ISO8879] [p.353] are expected to recognize a number of features that aren't widely supported by HTML user agents. We recommend that authors avoid using all of these features.

B.3.4 Boolean attributes

Authors should be aware that many user agents only recognize the minimized form of boolean attributes and not the full form.

For instance, authors may want to specify:

```
<OPTION selected>
instead of
<OPTION selected="selected">
```

B.3.5 Marked Sections

Marked sections play a role similar to the #ifdef construct recognized by C preprocessors.

```
<![INCLUDE[
<!-- this will be included -->
]]>
```

```
<![IGNORE[
<!-- this will be ignored -->
]]>
```

SGML also defines the use of marked sections for CDATA content, within which "<" is not treated as the start of a tag, e.g.,

```
<![CDATA[
<an> example of <sgml> markup that is
not <painful> to write with < and such.
]]>
```

The telltale sign that a user agent doesn't recognize a marked section is the appearance of "]]>", which is seen when the user agent mistakenly uses the first ">" character as the end of the tag starting with "<|[".

B.3.6 Processing Instructions

Processing instructions are a mechanism to capture platform-specific idioms. A processing instruction begins with <? and ends with >

```
<?instruction >
```

For example:

```
<?>
<?style tt = font courier>
<?page break>
<?experiment> ... <?/experiment>
```

Authors should be aware that many user agents render processing instructions as part of the document's text.

B.3.7 Shorthand markup

Some SGML SHORTTAG constructs save typing but add no expressive capability to the SGML application. Although these constructs technically introduce no ambiguity, they reduce the robustness of documents, especially when the language is enhanced to include new elements. Thus, while SHORTTAG constructs of SGML related to attributes are widely used and implemented, those related to elements are not. Documents that use them are conforming SGML documents, but are unlikely to work with many existing HTML tools.

The SHORTTAG constructs in question are the following:

- NET tags:

```
<name/.../
```

- closed Start Tag:

```
<name1<name2>
```

- Empty Start Tag:

```
<>
```

- Empty End Tag:

```
</>
```

B.4 Notes on helping search engines index your Web site

This section provides some simple suggestions that will make your documents more accessible to search engines.

Define the document language

In the global context of the Web it is important to know which human language a page was written in. This is discussed in the section on language information [p.79].

Specify language variants of this document

If you have prepared translations of this document into other languages, you should use the LINK element to reference these. This allows an indexing engine to offer users search results in the user's preferred language, regardless of how the query was written. For instance, the following links offer French and German alternatives to a search engine:

```
<LINK rel="alternate"
      type="text/html"
      href="mydoc-fr.html" hreflang="fr"
      lang="fr" title="La vie souteraine">
<LINK rel="alternate"
      type="text/html"
      href="mydoc-de.html" hreflang="de"
      lang="de" title="Das Leben im Untergrund">
```

Provide keywords and descriptions

Some indexing engines look for META elements that define a comma-separated list of keywords/phrases, or that give a short description. Search engines may present these keywords as the result of a search. The value of the name attribute sought by a search engine is not defined by this specification. Consider these examples,

```
<META name="keywords" content="vacation,Greece,sunshine">
<META name="description" content="Idyllic European vacations">
```

Indicate the beginning of a collection

Collections of word processing documents or presentations are frequently translated into collections of HTML documents. It is helpful for search results to reference the beginning of the collection in addition to the page hit by the search. You may help search engines by using the LINK element with **rel="start"** along with the title attribute, as in:

```
<LINK rel="start"
      type="text/html"
      href="page1.html"
      title="General Theory of Relativity">
```

Provide robots with indexing instructions

People may be surprised to find that their site has been indexed by an indexing robot and that the robot should not have been permitted to visit a sensitive part of the site. Many Web robots offer facilities for Web site administrators and content providers to limit what the robot does. This is achieved through two mechanisms: a "robots.txt" file and the META element in HTML documents, described below.

B.4.1 Search robots

The robots.txt file

When a Robot visits a Web site, say <http://www.foobar.com/>, it firsts checks for <http://www.foobar.com/robots.txt>. If it can find this document, it will analyze its contents to see if it is allowed to retrieve the document. You can customize the robots.txt file to apply only to specific robots, and to disallow access to specific directories or files.

Here is a sample robots.txt file that prevents all robots from visiting the entire site

```
User-agent: * # applies to all robots
Disallow: / # disallow indexing of all pages
```

The Robot will simply look for a "/robots.txt" URI on your site, where a site is defined as a HTTP server running on a particular host and port number. Here are some sample locations for robots.txt:

Site URI**URI for robots.txt**

`http://www.w3.org/` `http://www.w3.org/robots.txt`

`http://www.w3.org:80/` `http://www.w3.org:80/robots.txt`

`http://www.w3.org:1234/` `http://www.w3.org:1234/robots.txt`

`http://w3.org/` `http://w3.org/robots.txt`

There can only be a single `"/robots.txt"` on a site. Specifically, you should not put `"robots.txt"` files in user directories, because a robot will never look at them. If you want your users to be able to create their own `"robots.txt"`, you will need to merge them all into a single `"/robots.txt"`. If you don't want to do this your users might want to use the Robots META Tag instead.

Some tips: URI's are case-sensitive, and `"/robots.txt"` string must be all lower-case. Blank lines are not permitted within a single record in the `"robots.txt"` file. There must be exactly one `"User-agent"` field per record. The robot should be liberal in interpreting this field. A case-insensitive substring match of the name without version information is recommended.

If the value is `"*"`, the record describes the default access policy for any robot that has not matched any of the other records. It is not allowed to have multiple such records in the `"/robots.txt"` file.

The `"Disallow"` field specifies a partial URI that is not to be visited. This can be a full path, or a partial path; any URI that starts with this value will not be retrieved. For example,

```
Disallow: /help disallows both /help.html and /help/index.html, whereas
Disallow: /help/ would disallow /help/index.html but allow /help.html.
```

An empty value for `"Disallow"`, indicates that all URIs can be retrieved. At least one `"Disallow"` field must be present in the `robots.txt` file.

Robots and the META element

The `META` element allows HTML authors to tell visiting robots whether a document may be indexed, or used to harvest more links. No server administrator action is required.

In the following example a robot should neither index this document, nor analyze it for links.

```
<META name="ROBOTS" content="NOINDEX, NOFOLLOW" >
```

The list of terms in the content is `ALL`, `INDEX`, `NOFOLLOW`, `NOINDEX`.

Note. *In early 1997 only a few robots implement this, but this is expected to change as more public attention is given to controlling indexing robots.*

B.5 Notes on tables

B.5.1 Design rationale

The HTML table model has evolved from studies of existing SGML tables models, the treatment of tables in common word processing packages, and a wide range of tabular layout techniques in magazines, books and other paper-based documents. The model was chosen to allow simple tables to be expressed simply with extra complexity available when needed. This makes it practical to create the markup for HTML tables with everyday text editors and reduces the learning curve for getting started. This feature has been very important to the success of HTML to date.

Increasingly, people are creating tables by converting from other document formats or by creating them directly with WYSIWYG editors. It is important that the HTML table model fit well with these authoring tools. This affects how the cells that span multiple rows or columns are represented, and how alignment and other presentation properties are associated with groups of cells.

Dynamic reformatting

A major consideration for the HTML table model is that the author does not control how a user will size a table, what fonts he or she will use, etc. This makes it risky to rely on column widths specified in terms of absolute pixel units. Instead, tables must be able to change sizes dynamically to match the current window size and fonts. Authors can provide guidance as to the relative widths of columns, but user agents should ensure that columns are wide enough to render the width of the largest element of the cell's content. If the author's specification must be overridden, relative widths of individual columns should not be changed drastically.

Incremental display

For large tables or slow network connections, incremental table display is important to user satisfaction. User agents should be able to begin displaying a table before all of the data has been received. The default window width for most user agents shows about 80 characters, and the graphics for many HTML pages are designed with these defaults in mind. By specifying the number of columns, and including provision for control of table width and the widths of different columns, authors can give hints to user agents that allow the incremental display of table contents.

For incremental display, the browser needs the number of columns and their widths. The default width of the table is the current window size (`width="100%"`). This can be altered by setting the `width` attribute of the `TABLE` element. By default,

all columns have the same width, but you can specify column widths with one or more `COL` elements before the table data starts.

The remaining issue is the number of columns. Some people have suggested waiting until the first row of the table has been received, but this could take a long time if the cells have a lot of content. On the whole it makes more sense, when incremental display is desired, to get authors to explicitly specify the number of columns in the `TABLE` element.

Authors still need a way of telling user agents whether to use incremental display or to size the table automatically to fit the cell contents. In the two pass auto-sizing mode, the number of columns is determined by the first pass. In the incremental mode, the number of columns must be stated up front (with `COL` or `COLGROUP` elements).

Structure and presentation

HTML distinguishes structural markup such as paragraphs and quotations from rendering idioms such as margins, fonts, colors, etc. How does this distinction affect tables? From the purist's point of view, the alignment of text within table cells and the borders between cells is a rendering issue, not one of structure. In practice, though, it is useful to group these with the structural information, as these features are highly portable from one application to the next. The HTML table model leaves most rendering information to associated style sheets. The model presented in this specification is designed to take advantage of such style sheets but not to require them.

Current desktop publishing packages provide very rich control over the rendering of tables, and it would be impractical to reproduce this in HTML, without making HTML into a bulky rich text format like RTF or MIF. This specification does, however, offer authors the ability to choose from a set of commonly used classes of border styles. The `FRAME` attribute controls the appearance of the border frame around the table while the `RULES` attribute determines the choice of rulings within the table. A finer level of control will be supported via rendering annotations. The `STYLE` attribute can be used for specifying rendering information for individual elements. Further rendering information can be given with the `STYLE` element in the document head or via linked style sheets.

During the development of this specification, a number of avenues were investigated for specifying the ruling patterns for tables. One issue concerns the kinds of statements that can be made. Including support for edge subtraction as well as edge addition leads to relatively complex algorithms. For instance, work on allowing the full set of table elements to include the `FRAME` and `RULES` attributes led to an algorithm involving some 24 steps to determine whether a particular edge of a cell should be ruled or not. Even this additional complexity doesn't provide enough rendering control to meet the full range of needs for tables. The current specification deliberately sticks to a simple intuitive model, sufficient for most purposes. Further experimental work is needed before a more complex approach is standardized.

Row and column groups

This specification provides a superset of the simpler model presented in earlier work on HTML+. Tables are considered as being formed from an optional caption together with a sequence of rows, which in turn consist of a sequence of table cells. The model further differentiates header and data cells, and allows cells to span multiple rows and columns.

Following the CALS table model (see [CALS] [p.355]), this specification allows table rows to be grouped into head and body and foot sections. This simplifies the representation of rendering information and can be used to repeat table head and foot rows when breaking tables across page boundaries, or to provide fixed headers above a scrollable body panel. In the markup, the foot section is placed before the body sections. This is an optimization shared with CALS for dealing with very long tables. It allows the foot to be rendered without having to wait for the entire table to be processed.

Accessibility

For the visually impaired, HTML offers the hope of setting to rights the damage caused by the adoption of windows based graphical user interfaces. The HTML table model includes attributes for labeling each cell, to support high quality text to speech conversion. The same attributes can also be used to support automated import and export of table data to databases or spreadsheets.

B.5.2 Recommended Layout Algorithms

If `COL` or `COLGROUP` elements are present, they specify the number of columns and the table may be rendered using a fixed layout. Otherwise the autolayout algorithm described below should be used.

If the `width` attribute is not specified, visual user agents should assume a default value of 100% for formatting.

It is recommended that user agents increase table widths beyond the value specified by `width` in cases when cell contents would otherwise overflow. User agents that override the specified width should do so within reason. User agents may elect to split words across lines to avoid the need for excessive horizontal scrolling or when such scrolling is impractical or undesired.

For the purposes of layout, user agents should consider that table captions (specified by the `CAPTION` element) behave like cells. Each caption is a cell that spans all of the table's columns if at the top or bottom of the table, and rows if at the left or right side of the table.

Fixed Layout Algorithm

For this algorithm, it is assumed that the number of columns is known. The column widths by default should be set to the same size. Authors may override this by specifying relative or absolute column widths, using the `COLGROUP` or `COL` elements. The default table width is the space between the current left and right margins, but may be overridden by the `width` attribute on the `TABLE` element, or determined from absolute column widths. To deal with mixtures of absolute and relative column widths, the first step is to allocate space from the table width to columns with absolute widths. After this, the space remaining is divided up between the columns with relative widths.

The table syntax alone is insufficient to guarantee the consistency of attribute values. For instance, the number of `COL` and `COLGROUP` elements may be inconsistent with the number of columns implied by the table cells. A further problem occurs when the columns are too narrow to avoid overflow of cell contents. The width of the table as specified by the `TABLE` element or `COL` elements may result in overflow of cell contents. It is recommended that user agents attempt to recover gracefully from these situations, e.g., by hyphenating words [p.96] and resorting to splitting words if hyphenation points are unknown.

In the event that an indivisible element causes cell overflow, the user agent may consider adjusting column widths and re-rendering the table. In the worst case, clipping may be considered if column width adjustments and/or scrollable cell content are not feasible. In any case, if cell content is split or clipped this should be indicated to the user in an appropriate manner.

Autolayout Algorithm

If the number of columns is not specified with `COL` and `COLGROUP` elements, then the user agent should use the following autolayout algorithm. It uses two passes through the table data and scales linearly with the size of the table.

In the first pass, line wrapping is disabled, and the user agent keeps track of the minimum and maximum width of each cell. The maximum width is given by the widest line. Since line wrap has been disabled, paragraphs are treated as long lines unless broken by `BR` elements. The minimum width is given by the widest text element (word, image, etc.) taking into account leading indents and list bullets, etc. In other words, it is necessary to determine the minimum width a cell would require in a window of its own before the cell begins to overflow. Allowing user agents to split words will minimize the need for horizontal scrolling or in the worst case, clipping the cell contents.

This process also applies to any nested tables occurring in cell content. The minimum and maximum widths for cells in nested tables are used to determine the minimum and maximum widths for these tables and hence for the parent table cell itself. The algorithm is linear with aggregate cell content, and broadly speaking, independent of the depth of nesting.

To cope with character alignment of cell contents, the algorithm keeps three running min/max totals for each column: Left of align char, right of align char and unaligned. The minimum width for a column is then: `max(min_left + min_right, min_non-aligned)`.

The minimum and maximum cell widths are then used to determine the corresponding minimum and maximum widths for the columns. These in turn, are used to find the minimum and maximum width for the table. Note that cells can contain nested tables, but this doesn't complicate the code significantly. The next step is to assign column widths according to the available space (i.e., the space between the current left and right margins).

For cells that span multiple columns, a simple approach consists of apportioning the min/max widths evenly to each of the constituent columns. A slightly more complex approach is to use the min/max widths of unspanned cells to weight how spanned widths are apportioned. Experiments suggest that a blend of the two approaches gives good results for a wide range of tables.

The table borders and intercell margins need to be included in assigning column widths. There are three cases:

1. **The minimum table width is equal to or wider than the available space.** In this case, assign the minimum widths and allow the user to scroll horizontally. For conversion to braille, it will be necessary to replace the cells by references to notes containing their full content. By convention these appear before the table.
2. **The maximum table width fits within the available space.** In this case, set the columns to their maximum widths.
3. **The maximum width of the table is greater than the available space, but the minimum table width is smaller.** In this case, find the difference between the available space and the minimum table width, lets call it **W**. Lets also call **D** the difference between maximum and minimum width of the table.

For each column, let **d** be the difference between maximum and minimum width of that column. Now set the column's width to the minimum width plus **d** times **W** over **D**. This makes columns with large differences between minimum and maximum widths wider than columns with smaller differences.

This assignment step is then repeated for nested tables using the minimum and maximum widths derived for all such tables in the first pass. In this case, the width of the parent table cell plays the role of the current window size in the above description. This process is repeated recursively for all nested tables. The topmost table is then rendered using the assigned widths. Nested tables are subsequently rendered as part of the parent table's cell contents.

If the table width is specified with the `width` attribute, the user agent attempts to set column widths to match. The `width` attribute is not binding if this results in columns having less than their minimum (i.e., indivisible) widths.

If relative widths are specified with the `COL` element, the algorithm is modified to increase column widths over the minimum width to meet the relative width constraints. The `COL` elements should be taken as hints only, so columns shouldn't be set to less than their minimum width. Similarly, columns shouldn't be made so wide that the table stretches well beyond the extent of the window. If a `COL` element specifies a relative width of zero, the column should always be set to its minimum width.

When using the two pass layout algorithm, the default alignment position in the absence of an explicit or inherited `charoff` attribute can be determined by choosing the position that would center lines for which the widths before and after the alignment character are at the maximum values for any of the lines in the column for which `align="char"`. For incremental table layout the suggested default is `charoff="50%"`. If several cells in different rows for the same column use character alignment, then by default, all such cells should line up, regardless of which character is used for alignment. Rules for handling objects too large for a column apply when the explicit or implied alignment results in a situation where the data exceeds the assigned width of the column.

Choice of attribute names. *It would have been preferable to choose values for the `frame` attribute consistent with the `rules` attribute and the values used for alignment. For instance: `none`, `top`, `bottom`, `topbot`, `left`, `right`, `leftright`, `all`. Unfortunately, SGML requires enumerated attribute values to be unique for each element, independent of the attribute name. This causes immediate problems for "none", "left", "right" and "all". The values for the `frame` attribute have been chosen to avoid clashes with the `rules`, `align`, and `valign` attributes. This provides a measure of future proofing, as it is anticipated that the `frame` and `rules` attributes will be added to other table elements in future revisions to this specification. An alternative would be to make `frame` a CDATA attribute. The consensus of the W3C HTML Working Group was that the benefits of being able to use SGML validation tools to check attributes based on enumerated values outweighs the need for consistent names.*

B.6 Notes on forms

B.6.1 Incremental display

The incremental display of documents being received from the network gives rise to certain problems with respect to forms. User agents should prevent forms from being submitted until all of the form's elements have been received.

The incremental display of documents raises some issues with respect to tabbing navigation. The heuristic of giving focus to the lowest valued `tabindex` in the document seems reasonable enough at first glance. However this implies having to wait until all of the document's text is received, since until then, the lowest valued `tabindex` may still change. If the user hits the tab key before then, it is reasonable for user agents to move the focus to the lowest currently available `tabindex`.

If forms are associated with client-side scripts, there is further potential for problems. For instance, a script handler for a given field may refer to a field that doesn't yet exist.

B.6.2 Future projects

This specification defines a set of elements and attributes powerful enough to fulfill the general need for producing forms. However there is still room for many possible improvements. For instance the following problems could be addressed in the future:

- The range of form field types is too limited in comparison with modern user interfaces. For instance there is no provision for tabular data entry, sliders or multiple page layouts.
- Servers cannot update the fields in a submitted form and instead have to send a complete HTML document causing screen flicker.
- These also cause problems for speech based browsers, making it difficult for the visually impaired to interact with HTML forms.

Another possible extension would be to add the `usemap` attribute to `INPUT` for use as client-side image map when "`type=image`". The `AREA` element corresponding to the location clicked would contribute the value to be passed to the server. To avoid the need to modify server scripts, it may be appropriate to extend `AREA` to provide `x` and `y` values for use with the `INPUT` element.

B.7 Notes on scripting

B.7.1 Reserved syntax for future script macros

This specification reserves syntax for the future support of script macros in HTML CDATA attributes. The intention is to allow attributes to be set depending on the properties of objects that appear earlier on the page. The syntax is:

```
attribute = "... &{ macro_body }; ..."
```

Current Practice for Script Macros

The macro body is made up of one or more statements in the default scripting language (as per intrinsic event attributes). The semicolon following the right brace is always needed, as otherwise the right brace character "]" is treated as being part of the macro body. Its also worth noting that quote marks are always needed for attributes containing script macros.

The processing of CDATA attributes proceeds as follows:

1. The SGML parser evaluates any SGML entities (e.g., ">").
2. Next the script macros are evaluated by the script engine.
3. Finally the resultant character string is passed to the application for subsequent processing.

Macro processing takes place when the document is loaded (or reloaded) but does not take place again when the document is resized, repainted, etc.

DEPRECATED EXAMPLE:

Here are some examples using JavaScript. The first one randomizes the document background color:

```
<BODY bgcolor='&{randomrgb}'; />
```

Perhaps you want to dim the background for evening viewing:

```
<BODY bgcolor='&{if(Date.getHours > 18)...}'; />
```

The next example uses JavaScript to set the coordinates for a client-side image map:

```
<MAP NAME=foo>
  <AREA shape="rect" coords="&{myrect(imageuri)};" href="&{myuri};" alt=" " >
</MAP>
```

This example sets the size of an image based upon document properties:

```
<IMG src="bar.gif" width='&{document.banner.width/2}'; height='50%' alt="banner">
```

You can set the URI for a link or image by script:

```
<SCRIPT type="text/javascript">
  function manufacturer(widget) {
    ...
  }
  function location(manufacturer) {
    ...
  }
  function logo(manufacturer) {
    ...
  }
</SCRIPT>
<A href='&{location(manufacturer("widget"))}';>widget</A>
<IMG src='&{logo(manufacturer("widget"))}'; alt="Logo">
```

This last example shows how SGML CDATA attributes can be quoted using single or double quote marks. If you use single quotes around the attribute string then you can include double quote marks as part of the attribute string. Another approach is use `"` for double quote marks:

```
<IMG src='&{logo(manufacturer("&quot;widget&quot;"))}'; alt="Logo">
```

B.8 Notes on frames

Since there is no guarantee that a frame target name is unique, it is appropriate to describe the current practice in finding a frame given a target name:

1. If the target name is a reserved word as described in the normative text, apply it as described.
2. Otherwise, perform a depth-first search of the frame hierarchy in the window that contained the link. Use the first frame whose name is an exact match.
3. If no such frame was found in (2), apply step 2 to each window, in a front-to-back ordering. Stop as soon as you encounter a frame with exactly the same name.
4. If no such frame was found in (3), create a new window and assign it the target name.

B.9 Notes on accessibility

The W3C Web Accessibility Initiative ([WAI] [p.357]) is producing a series of guidelines to improve Web accessibility for people with disabilities. There are three sets of guidelines:

- Web Content Accessibility Guidelines ([WCGL] [p.357]), for authors and site managers. Please consult the Web Content Accessibility Guidelines for information about supplying alternative text for images, applets, scripts, etc.
- User Agent Accessibility Guidelines ([UAGL] [p.357]), for user agent developers (browsers, multimedia players, assistive technologies). Please consult these guidelines for guidance on handling alternate text.
- Authoring Tool Accessibility Guidelines ([ATGL] [p.355]), for authoring tool developers.

B.10 Notes on security

anchors, embedded images, and all other elements that contain URIs [p.51] as parameters may cause the URI to be dereferenced in response to user input. In this case, the security issues of [RFC1738] [p.354], section 6, should be considered. The widely deployed methods for submitting form requests -- HTTP and SMTP -- provide little assurance of confidentiality. Information providers who request sensitive information via forms -- especially with the `INPUT` element, `type="password"` -- should be aware and make their users aware of the lack of confidentiality.

B.10.1 Security issues for forms

A user agent should not send any file that the user has not explicitly asked to be sent. Thus, HTML user agents are expected to confirm any default file names that might be suggested by the `value` attribute of the `INPUT` element. Hidden controls must not specify files.

This specification does not contain a mechanism for encryption of the data; this should be handled by whatever other mechanisms are in place for secure transmission of data.

Once a file is uploaded, the processing agent should process and store it appropriately.

References

Contents

1. Normative references 353
2. Informative references 355

Normative references

- [CSS1]** "Cascading Style Sheets, level 1", H. W. Lie and B. Bos, 17 December 1996. Revised 11 January 1999. This document is <http://www.w3.org/TR/1999/REC-CSS1-19990111>
- [DATEIME]** "Date and Time Formats", W3C Note, M. Wolf and C. Wicksteed, 15 September 1997. Revised 27 August 1998. This document is <http://www.w3.org/TR/1998/NOTE-dateime-19980827>
- [HTML40]** "HTML 4.0 Specification", D. Raggett, A. Le Hors, I. Jacobs. The 24 April 1998 version is <http://www.w3.org/TR/1998/REC-html40-19980424>. The 24 April version included editorial changes from the original 18 December 1997 Revision.
- [IANA]** "Assigned Numbers", STD 2, RFC 1700, USC/ISI, J. Reynolds and J. Postel, October 1994.
- [ISO639]** "Codes for the representation of names of languages", ISO 639:1988. For more information, consult <http://www.iso.ch/cate/d4766.html>. Refer also to <http://www.oasis-open.org/cover/iso639a.html>.
- [ISO3166]** "Codes for the representation of names of countries", ISO 3166:1993.
- [ISO8601]** "Data elements and interchange formats -- Information interchange -- Representation of dates and times", ISO 8601:1988.
- [ISO8879]** "Information Processing -- Text and Office Systems -- Standard Generalized Markup Language (SGML)", ISO 8879:1986. Please consult <http://www.iso.ch/cate/d16387.html> for information about the standard.
- [ISO10646]** "Information Technology -- Universal Multiple-Octet Coded Character Set (UCS) -- Part 1: Architecture and Basic Multilingual Plane", ISO/IEC 10646-1:1993. This reference refers to a set of codepoints that may evolve as new characters are assigned to them. This reference therefore includes future amendments as long as they do **not** change character assignments up to and including the first five amendments to ISO/IEC 10646-1:1993. Also, this reference assumes that the character sets defined by ISO 10646 and Unicode remain

character-by-character equivalent. This reference also includes future publications of other parts of 10646 (i.e., other than Part 1) that define characters in planes 1-16.

- [ISO8591]** "Information Processing -- 8-bit single-byte coded graphic character sets -- Part 1: Latin alphabet No. 1", ISO 8859-1:1987.

[MIMETYPES]

List of registered content types (MIME types). Download a list of registered content types from <ftp://ftp.isi.edu/in-notes/iana/assignments/media-types/>.

- [RFC1555]** "Hebrew Character Encoding for Internet Messages", H. Nussbacher and Y. Bourvine, December 1993.

- [RFC1556]** "Handling of Bi-directional Texts in MIME", H. Nussbacher, December 1993.

- [RFC1738]** "Uniform Resource Locators", T. Berners-Lee, L. Masinter, and M. McCahill, December 1994.

- [RFC1766]** "Tags for the Identification of Languages", H. Alvestrand, March 1995. RFC1766 is expected to be updated by <http://www.ietf.org/internet-drafts/draft-alvestrand-lang-tags-v2-00.txt>, currently a work in progress.

- [RFC1808]** "Relative Uniform Resource Locators", R. Fielding, June 1995.

- [RFC2045]** "Multipurpose Internet Mail Extensions (MIME) Part One: Format of Internet Message Bodies", N. Freed and N. Borenstein, November 1996. Note that this RFC obsoletes RFC1521, RFC1522, and RFC1590.

- [RFC2046]** "Multipurpose Internet Mail Extensions (MIME) Part Two: Media Types", N. Freed and N. Borenstein, November 1996. Note that this RFC obsoletes RFC1521, RFC1522, and RFC1590.

- [RFC2119]** "Key words for use in RFCs to Indicate Requirement Levels", S. Bradner, March 1997.

- [RFC2141]** "URN Syntax", R. Moats, May 1997.

- [RFC2279]** "UTF-8, a transformation format of ISO 10646", F. Yergeau, January 1998. This RFC obsoletes RFC 2044.

- [RFC2616]** "Hypertext Transfer Protocol -- HTTP/1.1", R. Fielding, J. Gettys, J. Mogul, H. Frystyk Nielsen, L. Masinter, P. Leach and T. Berners-Lee, June 1999. This RFC obsoletes RFC 2068.

- [SRGB]** "A Standard Default color Space for the Internet", version 1.10, M. Stokes, M.

- Anderson, S. Chandrasekar, and R. Motta, 5 November 1996. This document is <http://www.w3.org/Graphics/Color/sRGB>
- [UNICODE]**
The Unicode Consortium. "The Unicode Standard, Version 3.0", Reading, MA, Addison-Wesley Developers Press, 2000. ISBN 0-201-61633-5. Refer also to <http://www.unicode.org/unicode/standard/versions/>.
- [URI]**
"Uniform Resource Identifiers (URI): Generic Syntax", T. Berners-Lee, R. Fielding, L. Masinter, August 1998. Note that RFC 2396 updates [RFC1738] [p.354] and [RFC1808] [p.354].
- [WEBSGML]**
"Final text of revised TC2 to ISO 8879:1986", C. F. Goldfarb, ed., 6 December 1998.

Informative references

- [ATGL]**
"Authoring Tool Accessibility Guidelines", J. Treviranus, J. Richards, I. Jacobs, C. McCathieNeville, eds. The latest Working Draft of these guidelines for designing accessible authoring tools is available at <http://www.w3.org/TR/WAI-AUTOOLS/>
- [BRYAN88]**
"SGML: An Author's Guide to the Standard Generalized Markup Language", M. Bryan, Addison-Wesley Publishing Co., 1988.
- [CALs]**
Continuous Acquisition and Life-Cycle Support (CALs). CALs is a Department of Defense strategy for achieving effective creation, exchange, and use of digital data for weapon systems and equipment. More information can be found on the CALs home page.
- [CHARSETS]**
Registered charset values. Download a list of registered charset values from <ftp://ftp.isi.edu/in-notes/iana/assignments/character-sets>.
- [CSS2]**
"Cascading Style Sheets, level 2", B. Bos, H. W. Lie, C. Lilley, and I. Jacobs, 12 May 1998. This document is <http://www.w3.org/TR/1998/REC-CSS2-19980512>
- [DCORE]**
The Dublin Core. For more information refer to <http://purl.org/dc>
- [ETHNO]**
"Ethnologue, Languages of the World", 12th Edition, Barbara F. Grimes editor, Summer Institute of Linguistics, October 1992.
- [GOLD90]**
"The SGML Handbook", C. F. Goldfarb, Clarendon Press, 1991.
- [HTML30]**
"HyperText Markup Language Specification Version 3.0", D. Raggett, September 1995. This document is <http://www.w3.org/MarkUp/html3/CoverPage>

- [HTML32]**
"HTML 3.2 Reference Specification", D. Raggett, 14 January 1997. This document is <http://www.w3.org/TR/REC-html32>
- [HTML3STYLE]**
"HTML and Style Sheets", B. Bos, D. Raggett, and H. Lie, 24 March 1997. This document is <http://www.w3.org/TR/WD-style-970324>
- [LEXHTML]**
"A Lexical Analyzer for HTML and Basic SGML", D. Connolly, 15 June 1996. This document is <http://www.w3.org/TR/WD-sgml-lex-960615>
- [OASISOPEN]**
The Organization for the Advancement of Structured Information Standards (OASIS): <http://www.oasis-open.org/>.
- [PICS]**
Platform for Internet Content (PICS). For more information refer to <http://www.w3.org/PICS/>
- [RDF10]**
"Resource Description Framework (RDF) Model and Syntax Specification", O. Lassila, R. Swick, eds., 22 February 1999. This document is <http://www.w3.org/TR/1999/REC-rdf-syntax-19990222>
- [RFC822]**
"Standard for the Format of ARPA Internet Text Messages", Revised by David H. Crocker, August 1982.
- [RFC850]**
"Standard for Interchange of USENET Messages", M. Horton, June 1983.
- [RFC1468]**
"Japanese Character Encoding for Internet Messages", J. Murai, M. Crispin, and E. van der Poel, June 1993.
- [RFC1630]**
"Universal Resource Identifiers in WWW: A Unifying Syntax for the Expression of Names and Addresses of Objects on the Network as used in the World-Wide Web", T. Berners-Lee, June 1994.
- [RFC1866]**
"HyperText Markup Language 2.0", T. Berners-Lee and D. Connolly, November 1995.
- [RFC1942]**
"HTML Tables", Dave Raggett, May 1996.
- [RFC2048]**
"Multipurpose Internet Mail Extensions (MIME) Part Four: Registration Procedures", N. Freed, J. Klensin, and J. Postel, November 1996. Note that this RFC obsoletes RFC1521, RFC1522, and RFC1590.
- [RFC2070]**
"Internationalization of the HyperText Markup Language", F. Yergeau, G. Nicol, G. Adams, and M. Dürst, January 1997.
- [RFC2388]**
"Returning Values from Forms: multipart/form-data", L. Masinter, August 1998. Refer also to RFC 1867, "Form-based File Upload in HTML", E. Nebel and L.

Masinter, November 1995.

[SP]

SP is a public domain SGML parser. Further information is available at <http://www.jclark.com/sp/index.htm>.

[SQ91]

"The SGML Primer", 3rd Edition, SoftQuad Inc., 1991.

[TAKADA]

"Multilingual Information Exchange through the World-Wide Web", Toshihiro Takada, Computer Networks and ISDN Systems, Vol. 27, No. 2, pp. 235-241, November 1994.

[UAGL]

"User Agent Accessibility Guidelines", J. Gunderson and I. Jacobs, eds. The latest Working Draft of these guidelines for designing accessible user agents is available at <http://www.w3.org/TR/WAI-USERAGENT>.

[WAI]

Guidelines for designing accessible HTML documents are available at the Web Accessibility Initiative (WAI) Web site: <http://www.w3.org/WAI/>.

[WCGL]

"Web Content Accessibility Guidelines 1.0", W. Chisholm, G. Vanderheiden, and I. Jacobs, eds., 5 May 1999. This document is <http://www.w3.org/TR/1999/WAI-WEBCONTENT-19990505>.

[VANH90]

"Practical SGML", E. van Herwijnen, Kluwer Academic Publishers Group, Norwell and Dordrecht, 1990.

[XHTML]

"XHTML[tm] 1.0: The Extensible HyperText Markup Language", S. Pemberton et al. The latest version of this specification is available at <http://www.w3.org/TR/xhtml1>. As of the publication of the current document, XHTML 1.0 is a W3C Proposed Recommendation.

Index of Elements

Legend: Optional, Forbidden, Empty, Deprecated, Loose DTD, Frameset DTD

Name	Start Tag	End Tag	Empty	Depr.	DTD	Description
A [p.149]						anchor
ABBR [p.90]						abbreviated form (e.g., WWW, HTTP, etc.)
ACRONYM [p.90]						
ADDRESS [p.76]						information on author
APPLET [p.171]				D	L	Java applet
AREA [p.174]		F	E			client-side image map area
B [p.199]						bold text style
BASE [p.157]		F	E			document base URI
BASEFONT [p.200]		F	E	D	L	base font size
BDO [p.85]						18N BiDi over-ride
BIG [p.199]						large text style
BLOCKQUOTE [p.92]						long quotation
BODY [p.69]	O	O				document body
BR [p.96]		F	E			forced line break
BUTTON [p.228]						push button
CAPTION [p.115]						table caption
CENTER [p.197]				D	L	shorthand for DIV align=center
CITE [p.90]						citation
CODE [p.90]						computer code fragment

COL [p.120]			F	E			table column
COLGROUP [p.118]			O				table column group
DD [p.106]			O				definition description
DEL [p.99]							deleted text
DFN [p.90]							instance definition
DIR [p.109]					D	L	directory list
DIV [p.73]							generic language/style container
DL [p.106]							definition list
DT [p.106]			O				definition term
EM [p.90]							emphasis
FIELDSET [p.239]							form control group
FONT [p.200]					D	L	local change to font
FORM [p.222]							interactive form
FRAME [p.209]			F	E		F	subwindow
FRAMESET [p.206]						F	window subdivision
H1 [p.75]							heading
H2 [p.75]							heading
H3 [p.75]							heading
H4 [p.75]							heading
H5 [p.75]							heading
H6 [p.75]							heading
HEAD [p.62]			O				document head
HR [p.202]			F	E			horizontal rule
HTML [p.61]			O				document root element
I [p.199]							italic text style
IFRAME [p.217]						L	inline subwindow

IMG [p.160]		F	E			Embedded image
INPUT [p.224]		F	E			form control
INS [p.99]						inserted text
ISINDEX [p.236]		F	E	D	L	single line prompt
KBD [p.90]						text to be entered by the user
LABEL [p.237]						form field label text
LEGEND [p.239]						fieldset legend
LI [p.105]		O				list item
LINK [p.154]		F	E			a media-independent link
MAP [p.174]						client-side image map
MENU [p.109]				D	L	menu list
META [p.65]		F	E			generic metainformation
NOFRAMES [p.214]					F	alternate content container for non frame-based rendering
NOSCRIPT [p.258]						alternate content container for non script-based rendering
OBJECT [p.162]						generic embedded object
OL [p.104]						ordered list
OPTGROUP [p.230]						option group
OPTION [p.230]		O				selectable choice
P [p.95]		O				paragraph
PARAM [p.167]		F	E			named property value
PRE [p.97]						preformatted text
Q [p.92]						short inline quotation
S [p.199]				D	L	strike-through text style
SAMP [p.90]						sample program output, scripts, etc.

SCRIPT [p.252]						script statements
SELECT [p.230]						option selector
SMALL [p.199]						small text style
SPAN [p.73]						generic language/style container
STRIKE [p.199]			D	L		strike-through text
STRONG [p.90]						strong emphasis
STYLE [p.187]						style info
SUB [p.94]						subscript
SUP [p.94]						superscript
TABLE [p.113]						
TBODY [p.116]		O				table body
TD [p.125]				O		table data cell
TEXTAREA [p.234]						multi-line text field
TFOOT [p.116]				O		table footer
TH [p.125]				O		table header cell
THEAD [p.116]				O		table header
TITLE [p.62]						document title
TR [p.124]				O		table row
TT [p.199]						teletype or monospaced text style
U [p.199]			D	L		underlined text style
UL [p.104]						unordered list
VAR [p.90]						instance of a variable or program argument

Index of Attributes

Legend: *Deprecated*, *Loose DTD*, *Frameset DTD*

Name	Related Elements	Type	Default	Depr.	DTD	Comment
abbr [p.126]	TD, TH	%Text; [p.266]	#IMPLIED			abbreviation for header cell
accept-charset [p.223]	FORM	%Charsets; [p.266]	#IMPLIED			list of supported charsets
accept [p.223]	FORM, INPUT	%ContentTypes; [p.266]	#IMPLIED			list of MIME types for file upload
accesskey [p.242]	A, AREA, BUTTON, INPUT, LABEL, LEGEND, TEXTAREA	%Character; [p.266]	#IMPLIED			accessibility key character
action [p.223]	FORM	%URI; [p.266]	#REQUIRED			server-side form handler
align [p.115]	CAPTION	%CAAlign; [p.292]	#IMPLIED	D	L	relative to table
align [p.180]	APPLET, IFRAME, IMG, INPUT, OBJECT	%IAAlign; [p.285]	#IMPLIED	D	L	vertical or horizontal alignment
align [p.239]	LEGEND	%LAAlign; [p.290]	#IMPLIED	D	L	relative to fieldset
align [p.113]	TABLE	%TAAlign; [p.291]	#IMPLIED	D	L	table position relative to window
align [p.202]	HR	(left center right)	#IMPLIED	D	L	
align [p.196]	DIV, H1, H2, H3, H4, H5, H6, P	(left center right justify)	#IMPLIED	D	L	align, text alignment
align [p.132]	COLGROUP, TBODY, TD, TFOOT, TH, THEAD, TR	(left center right justify char)	#IMPLIED			
alink [p.69]	BODY	%Color; [p.281]	#IMPLIED	D	L	color of selected links
alt [p.181]	APPLET	%Text; [p.281]	#IMPLIED	D	L	short description
alt [p.181]	AREA, IMG	%Text; [p.266]	#REQUIRED			short description
alt [p.181]	INPUT	CDATA [p.50]	#IMPLIED			short description
archive [p.171]	APPLET	CDATA [p.50]	#IMPLIED	D	L	comma-separated archive list
archive [p.163]	OBJECT	CDATA [p.50]	#IMPLIED			space-separated list of URIs

axis [p.126]	TD, TH	CDATA [p.50]	#IMPLIED		comma-separated list of related headers
background [p.69]	BODY	%URI; [p.280]	#IMPLIED	D	texture tile for document background
bgcolor [p.195]	TABLE	%Color; [p.281]	#IMPLIED	D	background color for cells
bgcolor [p.195]	TR	%Color; [p.281]	#IMPLIED	D	background color for row
bgcolor [p.195]	TD, TH	%Color; [p.281]	#IMPLIED	D	cell background color
bgcolor [p.195]	BODY	%Color; [p.281]	#IMPLIED	D	document background color
border [p.130]	TABLE	%Pixels; [p.270]	#IMPLIED		controls frame width around table
border [p.180]	IMG, OBJECT	%Pixels; [p.285]	#IMPLIED	D	link border width
cellpadding [p.135]	TABLE	%Length; [p.270]	#IMPLIED		spacing within cells
cellspacing [p.134]	TABLE	%Length; [p.270]	#IMPLIED		spacing between cells
char [p.132]	COL, COLGROUP, TBODY, TD, TFOOT, TH, THEAD, TR	%Character; [p.266]	#IMPLIED		alignment char, e.g. char=';
charoff [p.133]	COL, COLGROUP, TBODY, TD, TFOOT, TH, THEAD, TR	%Length; [p.270]	#IMPLIED		offset for alignment char
charset [p.150]	A, LINK, SCRIPT	%Charset; [p.266]	#IMPLIED		char encoding of linked resource
checked [p.225]	INPUT	(checked)	#IMPLIED		for radio buttons and check boxes
cite [p.93]	BLOCKQUOTE, Q	%URI; [p.266]	#IMPLIED		URI for source document or msg
cite [p.100]	DEL, INS	%URI; [p.266]	#IMPLIED		info on reason for change
class [p.71]	All elements but BASE, BASEFONT, HEAD, HTML, META, PARAM, SCRIPT, STYLE, TITLE	CDATA [p.50]	#IMPLIED		space-separated list of classes

classid [p.163]	OBJECT	%URI; [p.266]	#IMPLIED		identifies an implementation
clear [p.198]	BR	(left all right none)	none	D L	control of text flow
code [p.171]	APPLET	CDATA [p.50]	#IMPLIED	D L	applet class file
codebase [p.163]	OBJECT	%URI; [p.266]	#IMPLIED		base URI for classid, data, archive
codebase [p.171]	APPLET	%URI; [p.280]	#IMPLIED	D L	optional base URI for applet
codetype [p.163]	OBJECT	%ContentType; [p.266]	#IMPLIED		content type for code
color [p.201]	BASEFONT, FONT	%Color; [p.281]	#IMPLIED	D L	text color
cols [p.207]	FRAMESET	%MultiLengths; [p.270]	#IMPLIED	F	list of lengths, default: 100% (1 col)
cols [p.235]	TEXTAREA	NUMBER [p.50]	#REQUIRED		
colspan [p.126]	TD, TH	NUMBER [p.50]	1		number of cols spanned by cell
compact [p.105]	DIR, DL, MENU, OL, UL	(compact)	#IMPLIED	D L	reduced interitem spacing
content [p.65]	META	CDATA [p.50]	#REQUIRED		associated information
coords [p.175]	AREA	%Coords; [p.269]	#IMPLIED		comma-separated list of lengths
coords [p.175]	A	%Coords; [p.269]	#IMPLIED		for use with client-side image maps
data [p.163]	OBJECT	%URI; [p.266]	#IMPLIED		reference to object's data
datetime [p.100]	DEL, INS	%Datetime; [p.266]	#IMPLIED		date and time of change
declare [p.164]	OBJECT	(declare)	#IMPLIED		declare but don't instantiate flag
defer [p.252]	SCRIPT	(defer)	#IMPLIED		UA may defer execution of script
dir [p.82]	All elements but APPLET, BASE, BASEFONT, BDO, BR, FRAME, FRAMESET, IFRAME, PARAM, SCRIPT	(ltr rtl)	#IMPLIED		direction for weak/neutral text

dir [p.85]	BDO BUTTON, INPUT, OPTGROUP, OPTION, SELECT, TEXTAREA	(ltr rtl)	#REQUIRED		directionality
disabled [p.244]		(disabled)	#IMPLIED		unavailable in this context
enctype [p.223]	FORM	%ContentType; [p.266]	"application/x-www-form-urlencoded"		
face [p.201]	BASEFONT, FONT	CDATA [p.50]	#IMPLIED	D L	comma-separated list of font names
for [p.237]	LABEL	IDREF [p.50]	#IMPLIED		matches field ID value
frame [p.130]	TABLE	%TFrame; [p.275]	#IMPLIED		which parts of frame to render
frameborder [p.210]	FRAME, IFRAME	(1 0)	1	F	request frame borders?
headers [p.125]	TD, TH	IDREFS [p.50]	#IMPLIED		list of id's for header cells
height [p.217]	IFRAME	%Length; [p.285]	#IMPLIED	L	frame height
height [p.126]	TD, TH	%Length; [p.285]	#IMPLIED	D L	height for cell
height [p.179]	IMG, OBJECT	%Length; [p.270]	#IMPLIED		override height
height [p.172]	APPLET	%Length; [p.285]	#REQUIRED	D L	initial height
href [p.149]	A, AREA, LINK	%URI; [p.266]	#IMPLIED		URI for linked resource
href [p.157]	BASE	%URI; [p.266]	#IMPLIED		URI that acts as base URI
hreflang [p.149]	A, LINK	%LanguageCode; [p.266]	#IMPLIED		language code
hspace [p.180]	APPLET, IMG, OBJECT	%Pixels; [p.285]	#IMPLIED	D L	horizontal gutter
http-equiv [p.65]	META	NAME [p.50]	#IMPLIED		HTTP response header name
id [p.71]	All elements but BASE, HEAD, HTML, META, SCRIPT, STYLE, TITLE	ID [p.50]	#IMPLIED		document-wide unique id
ismap [p.179]	IMG, INPUT	(ismap)	#IMPLIED		use server-side image map
label [p.232]	OPTION	%Text; [p.266]	#IMPLIED		for use in hierarchical menus
label [p.232]	OPTGROUP	%Text; [p.266]	#REQUIRED		for use in hierarchical menus

lang [p.79]	All elements but APPLLET, BASE, BASEFONT, BR, FRAME, FRAMESET, IFRAME, PARAM, SCRIPT	%LanguageCode [p.266]	#IMPLIED		language code	
language [p.252]	SCRIPT	CDATA [p.50]	#IMPLIED	D	L	predefined script language name
link [p.69]	BODY	%Color; [p.281]	#IMPLIED	D	L	color of links
longdesc [p.161]	IMG	%URI; [p.266]	#IMPLIED			link to long description (complements alt)
longdesc [p.209]	FRAME, IFRAME	%URI; [p.266]	#IMPLIED		F	link to long description (complements title)
marginheight [p.210]	FRAME, IFRAME	%Pixels; [p.270]	#IMPLIED		F	margin height in pixels
marginwidth [p.210]	FRAME, IFRAME	%Pixels; [p.270]	#IMPLIED		F	margin widths in pixels
maxlength [p.225]	INPUT	NUMBER [p.50]	#IMPLIED			max chars for text fields
media [p.187]	STYLE	%MediaDesc; [p.266]	#IMPLIED			designed for use with these media
media [p.187]	LINK	%MediaDesc; [p.266]	#IMPLIED			for rendering on these media
method [p.223]	FORM	(GET POST)	GET			HTTP method used to submit the form
multiple [p.230]	SELECT	(multiple)	#IMPLIED			default is single selection
name [p.229]	BUTTON, TEXTAREA	CDATA [p.50]	#IMPLIED			
name [p.171]	APPLET	CDATA [p.50]	#IMPLIED	D	L	allows applets to find each other
name [p.230]	SELECT	CDATA [p.50]	#IMPLIED			field name
name [p.223]	FORM	CDATA [p.50]	#IMPLIED			name of form for scripting
name [p.209]	FRAME, IFRAME	CDATA [p.50]	#IMPLIED		F	name of frame for targeting
name [p.161]	IMG	CDATA [p.50]	#IMPLIED			name of image for scripting
name [p.149]	A	CDATA [p.50]	#IMPLIED			named link end

name [p.225]	INPUT, OBJECT	CDATA [p.50]	#IMPLIED			submit as part of form	
name [p.174]	MAP	CDATA [p.50]	#REQUIRED			for reference by usemap	
name [p.167]	PARAM	CDATA [p.50]	#REQUIRED			property name	
name [p.65]	META	NAME [p.50]	#IMPLIED			metainformation name	
nohref [p.175]	AREA	(nohref)	#IMPLIED			this region has no action	
noresize [p.210]	FRAME	(noresize)	#IMPLIED		F	allow users to resize frames?	
noshade [p.202]	HR	(noshade)	#IMPLIED		D	L	
nowrap [p.126]	TD, TH	(nowrap)	#IMPLIED		D	L	suppress word wrap
object [p.171]	APPLET	CDATA [p.50]	#IMPLIED		D	L	serialized applet file
onblur [p.255]	A, AREA, BUTTON, INPUT, LABEL, SELECT, TEXTAREA	%Script; [p.266]	#IMPLIED			the element lost the focus	
onchange [p.256]	INPUT, SELECT, TEXTAREA	%Script; [p.266]	#IMPLIED			the element value was changed	
onclick [p.255]	All elements but APPLLET, BASE, BASEFONT, BDO, BR, FONT, FRAME, FRAMESET, HEAD, HTML, IFRAME, ISINDEX, META, PARAM, SCRIPT, STYLE, TITLE	%Script; [p.266]	#IMPLIED			a pointer button was clicked	
ondblclick [p.255]	All elements but APPLLET, BASE, BASEFONT, BDO, BR, FONT, FRAME, FRAMESET, HEAD, HTML, IFRAME, ISINDEX, META, PARAM, SCRIPT, STYLE, TITLE	%Script; [p.266]	#IMPLIED			a pointer button was double clicked	

onfocus [p.255]	A, AREA, BUTTON, INPUT, LABEL, SELECT, TEXTAREA	%Script; [p.266]	#IMPLIED	the element got the focus
onkeydown [p.255]	All elements but APPLE, BASE, BASEFONT, BDO, BR, FONT, FRAME, FRAMESET, HEAD, HTML, IFRAME, ISINDEX, META, PARAM, SCRIPT, STYLE, TITLE	%Script; [p.266]	#IMPLIED	a key was pressed down
onkeypress [p.255]	All elements but APPLE, BASE, BASEFONT, BDO, BR, FONT, FRAME, FRAMESET, HEAD, HTML, IFRAME, ISINDEX, META, PARAM, SCRIPT, STYLE, TITLE	%Script; [p.266]	#IMPLIED	a key was pressed and released
onkeyup [p.255]	All elements but APPLE, BASE, BASEFONT, BDO, BR, FONT, FRAME, FRAMESET, HEAD, HTML, IFRAME, ISINDEX, META, PARAM, SCRIPT, STYLE, TITLE	%Script; [p.266]	#IMPLIED	a key was released
onload [p.254]	FRAMESET	%Script; [p.266]	#IMPLIED	F all the frames have been loaded
onload [p.254]	BODY	%Script; [p.266]	#IMPLIED	the document has been loaded

onmousedown [p.255]	All elements but APPLE, BASE, BASEFONT, BDO, BR, FONT, FRAME, FRAMESET, HEAD, HTML, IFRAME, ISINDEX, META, PARAM, SCRIPT, STYLE, TITLE	%Script; [p.266]	#IMPLIED	a pointer button was pressed down
onmouseout [p.255]	All elements but APPLE, BASE, BASEFONT, BDO, BR, FONT, FRAME, FRAMESET, HEAD, HTML, IFRAME, ISINDEX, META, PARAM, SCRIPT, STYLE, TITLE	%Script; [p.266]	#IMPLIED	a pointer was moved away
onmouseover [p.255]	All elements but APPLE, BASE, BASEFONT, BDO, BR, FONT, FRAME, FRAMESET, HEAD, HTML, IFRAME, ISINDEX, META, PARAM, SCRIPT, STYLE, TITLE	%Script; [p.266]	#IMPLIED	a pointer was moved onto

onmouseover [p.255]	All elements but APPLET, BASE, BASEFONT, BDO, BR, FONT, FRAME, FRAMESET, HEAD, HTML, IFRAME, ISINDEX, META, PARAM, SCRIPT, STYLE, TITLE	%Script; [p.266]	#IMPLIED	a pointer button was released
onreset [p.256]	FORM	%Script; [p.266]	#IMPLIED	the form was reset
onselect [p.256]	INPUT, TEXTAREA	%Script; [p.266]	#IMPLIED	some text was selected
onsubmit [p.255]	FORM	%Script; [p.266]	#IMPLIED	the form was submitted
onunload [p.255]	FRAMESET	%Script; [p.266]	#IMPLIED	all the frames have been removed
onunload [p.255]	BODY	%Script; [p.266]	#IMPLIED	the document has been removed
profile [p.62]	HEAD	%URI; [p.266]	#IMPLIED	named dictionary of meta info
prompt [p.236]	ISINDEX	%Text; [p.281]	#IMPLIED	prompt message
readonly [p.244]	TEXTAREA	(readonly)	#IMPLIED	
readonly [p.244]	INPUT	(readonly)	#IMPLIED	for text and passwd
rel [p.149]	A, LINK	%LinkTypes; [p.266]	#IMPLIED	forward link types
rev [p.150]	A, LINK	%LinkTypes; [p.266]	#IMPLIED	reverse link types
rows [p.207]	FRAMESET	%Multilengths; [p.270]	#IMPLIED	list of lengths, default: 100% (1 row)
rows [p.235]	TEXTAREA	NUMBER [p.50]	#REQUIRED	
rowspan [p.126]	TD, TH	NUMBER [p.50]	1	number of rows spanned by cell
rules [p.130]	TABLE	%TRules; [p.275]	#IMPLIED	rulings between rows and cols
scheme [p.65]	META	CDATA [p.50]	#IMPLIED	select form of content
scope [p.125]	TD, TH	%Scope; [p.277]	#IMPLIED	scope covered by header cells

scrolling [p.210]	FRAME, IFRAME	(yes no auto)	auto	F	scrollbar or none
selected [p.232]	OPTION	(selected)	#IMPLIED		
shape [p.174]	AREA	%Shape; [p.269]	rect		controls interpretation of coords
shape [p.174]	A	%Shape; [p.269]	rect		for use with client-side image maps
size [p.202]	HR	%Pixels; [p.285]	#IMPLIED	D	L
size [p.201]	FONT	CDATA [p.50]	#IMPLIED	D	L
size [p.225]	INPUT	CDATA [p.50]	#IMPLIED		specific to each type of field
size [p.201]	BASEFONT	CDATA [p.50]	#REQUIRED	D	L
size [p.230]	SELECT	NUMBER [p.50]	#IMPLIED		base font size for FONT elements
span [p.121]	COL	NUMBER [p.50]	1		rows visible
span [p.119]	COLGROUP	NUMBER [p.50]	1		COL attributes affect N columns
src [p.252]	SCRIPT	%URI; [p.266]	#IMPLIED		default number of columns in group
src [p.225]	INPUT	%URI; [p.266]	#IMPLIED		URI for an external script
src [p.210]	FRAME, IFRAME	%URI; [p.266]	#IMPLIED	F	for fields with images
src [p.161]	IMG	%URI; [p.266]	#REQUIRED		source of frame content
standby [p.164]	OBJECT	%Text; [p.266]	#IMPLIED		URI of image to embed
start [p.105]	OL	NUMBER [p.50]	#IMPLIED	D	L
style [p.186]	All elements but BASE, BASEFONT, HEAD, HTML, META, PARAM, SCRIPT, STYLE, TITLE	%StyleSheet; [p.266]	#IMPLIED		message to show while loading
summary [p.113]	TABLE	%Text; [p.266]	#IMPLIED		starting sequence number
					associated style info
					purpose/structure for speech output

tabindex [p.241]	A, AREA, BUTTON, INPUT, OBJECT, SELECT, TEXTAREA	NUMBER [p.50]	#IMPLIED	position in tabbing order
target [p.212]	A, AREA, BASE, FORM, LINK	%FrameTarget; [p.281]	#IMPLIED	render in this frame
text [p.69]	BODY	%Color; [p.281]	#IMPLIED	document text color
title [p.63]	All elements but BASE, BASEFONT, HEAD, HTML, META, PARAM, SCRIPT, TITLE	%Text; [p.266]	#IMPLIED	advisory title
type [p.149]	A, LINK	%ContentType; [p.266]	#IMPLIED	advisory content type
type [p.163]	OBJECT	%ContentType; [p.266]	#IMPLIED	content type for data
type [p.167]	PARAM	%ContentType; [p.266]	#IMPLIED	content type for value when valueType=ref
type [p.252]	SCRIPT	%ContentType; [p.266]	#REQUIRED	content type of script language
type [p.187]	STYLE	%ContentType; [p.266]	#REQUIRED	content type of style language
type [p.225]	INPUT	%InputType; [p.273]	TEXT	what kind of widget is needed
type [p.105]	LI	%ListStyle; [p.289]	#IMPLIED	list item style
type [p.105]	OL	%OLStyle; [p.288]	#IMPLIED	numbering style
type [p.105]	UL	%ULStyle; [p.288]	#IMPLIED	bullet style
type [p.229]	BUTTON	(button submit reset)	submit	for use as form button
usemap [p.175]	IMG, INPUT, OBJECT	%URI; [p.266]	#IMPLIED	use client-side image map
valign [p.132]	COL, COLGROUP, TBODY, TD, TFOOT, TH, THEAD, TR	(top middle bottom baseline)	#IMPLIED	vertical alignment in cells
value [p.225]	INPUT	CDATA [p.50]	#IMPLIED	Specify for radio buttons and checkboxes
value [p.232]	OPTION	CDATA [p.50]	#IMPLIED	defaults to element content

value [p.167]	PARAM	CDATA [p.50]	#IMPLIED	property value
value [p.229]	BUTTON	CDATA [p.50]	#IMPLIED	sent to server when submitted
value [p.105]	LI	NUMBER [p.50]	#IMPLIED	reset sequence number
valueType [p.167]	PARAM	(DATA REF OBJECT)	DATA	How to interpret value
version [p.61]	HTML	CDATA [p.50]	%HTMLVersion; [p.279]	Constant
vlink [p.69]	BODY	%Color; [p.281]	#IMPLIED	color of visited links
vspace [p.180]	APPLET, IMG, OBJECT	%Pixels; [p.285]	#IMPLIED	vertical gutter
width [p.202]	HR	%Length; [p.285]	#IMPLIED	
width [p.217]	IFRAME	%Length; [p.285]	#IMPLIED	frame width
width [p.179]	IMG, OBJECT	%Length; [p.270]	#IMPLIED	override width
width [p.113]	TABLE	%Length; [p.270]	#IMPLIED	table width
width [p.126]	TD, TH	%Length; [p.285]	#IMPLIED	width for cell
width [p.171]	APPLET	%Length; [p.285]	#REQUIRED	initial width
width [p.121]	COL	%MultiLength; [p.270]	#IMPLIED	column width specification
width [p.119]	COLGROUP	%MultiLength; [p.270]	#IMPLIED	default width for enclosed COLs
width [p.97]	PRE	NUMBER [p.50]	#IMPLIED	

Index

abbreviations and acronyms 92
 access key 242
 accessibility
 access keys 242
 alternate object content 166
 alternate text 181
 and alternate frame content 214
 and long frame descriptions 215
 and style sheets 184
 features in HTML 4 22
 long image description 162
 of image maps 174, 176
 alignment
 floating text 198
 floats 197
 of block-level elements 195
 of images 180
 of objects 180
 of table contents 132
 alternate style sheets **190**
 alternate text
 specifying 181
 anchor **145**
 ASCII characters in name 152
 case of name 152
 character references in name 153
 creation of with A 150
 creation with id attribute 152
 name space of 153
 non-ASCII characters in name 335
 set by script 150
 syntax of name 152
 uniqueness of name 152
 with A vs. id 153
 applet
 ways to include 160
 application/x-www-form-urlencoded **247**, 223
 attribute **29**
 #FIXED value of **34**
 #IMPLIED value of **34**
 #REQUIRED value of **34**
 boolean **36**
 case of values 34
 case-insensitive 30

declaration of in DTD 34
 minimized boolean **36**
 quotation marks around value 30
 %attrs; **35**
 author **37**
 authoring tool **37**, 90
 and default style sheet language 186
 background color 195
 base URI 157
 bidirection
 Unicode algorithm 82
 and character encoding 86
 and style sheets 88
 override of 85
 block-level
 and bidirection 83, 73
 element **73**
 %block; 32
 BODY
 none in frameset 71
 boolean attribute 337, **36**
 minimized **36**
 border
 around a frame 210
 around a table 130
 around image 180
 around object 180
 cascading style sheets 192
 case
 of URIs 51
 of anchor name 152
 of attribute names 30
 of attribute values 30, 34, 49
 of character encodings 53
 of character entity reference 46
 of color names 51
 of content types 53
 of element names 29
 of language codes 53
 of length values 52
 of link types 54
 of media descriptors 57
 of numeric character references 45
 of script data 57
 of style data 57

- catalog for HTML 262
- CDATA 34, 50
 - script and style data 50
- CERN 21
- character encoding 42, 40
 - UTF-1 43
 - UTF-16 43
 - and bidirection 86
 - choice of 42
 - common examples 43
 - default 44
 - for form submission 223
 - names of 53
 - of links 148
 - specification of 43
 - user agent's determination of 44
- character entity references 46
- character reference 45, 30
 - for directionality 87
- character repertoire 41
- %Character; 53
- characters
 - abstract 41
 - access key 242
 - best effort to render 80
 - handling undisplayable 47
 - rendering undisplayable 47
- %Charset; 53
- checkbox 221
- class attribute
 - roles of 71
- client-side image map 174
 - creation of 176
- clipping
 - table text 116
 - code position 41
- color
 - background 195
 - names of 51
- %Color; 51
- column
 - number of in a table 121
 - width of in a table 122
- column group 118
- comments
 - character references in 45

- in DTD 31
- in HTML 30
- informative only 39
- not rendered 39
- used to hide script data 259
- used to hide style sheet data 193
- conformance 38
- content model 32
 - excluded elements in 34
 - syntax of in DTD 33
- content type
 - application/x-www-form-urlencoded 247
 - multipart/form-data 248
 - text/html 39
- content types
 - for encoding form data 247
- Content-Language header 90
- Content-Script-Type header 253
- Content-Style-Type header 186
- Content-Type header 43
- %ContentType; 53
- control 220
 - access key for 242
 - control name 220
 - control value 220
 - disabled 244
 - events for 256
 - giving focus to 241
 - initial value 220
 - read only 244
 - successful 245
 - tabbing navigation 241
 - types of 221
- coordinates
 - of form submit click 226
 - of server-side image map 179
- data cell
 - in table 127
- data type
 - CDATA 50
 - ID 50
 - IDREF 50
 - IDREFS 50
 - NAME 50
 - NUMBER 50

- date
 - format of **54**
 - of inserted and deleted text **100**
 - %Datetime: **54**
- default
 - character encoding **44**
 - scripting language **253**
 - style sheet language **186**
 - target frame **213**
- deprecated **38**
- elements **327**
- direction
 - inheritance of for nested elements **83**
 - of table information **115**
 - of text **82**
- disabled controls **244**
- not successful **245**
- document
 - SGML validation **261**
 - dynamic modification with script **258**
 - ways to embed **173**
 - ways to include **217, 160**
- document character set **41**
- ISO10646 **41, 261**
- equivalence of ISO10646 and UNICODE **42**
- document type declaration **60**
- for frameset DTD **60**
- for strict DTD **60**
- for transitional DTD **60**
- document type definition **28**
- DTD fragments conform to **39**
- comments in **31**
- examples conform to **39**
- frameset **297**
- how to read **31**
- strict **265**
- transitional **279**
- Dublin Core **69**
- element
 - block-level **73**
 - case-insensitive **29**
 - content model of **32**
 - empty **29, 32**
 - end tag **28**
 - inline **73**
 - list of deprecated **327**

- list of obsolete **328**
- omitted end tag **28**
- omitted start tag **28**
- references from scripts **254**
- start tag **28**
- support for deprecated **38**
- support for obsolete **38**
- type declaration **28, 32**
- types **28**
- unique identifier for **71**
- empty element **32**
- end tag **28**
- declared as optional **32**
- omitted **28**
- entity sets
 - URLs for HTML 4.01 **60**
- error
 - handling by user agents **334, 38**
 - image map with IMG in BUTTON **230**
 - rendering style rules in STYLE **187**
 - unavailable resource **154**
- events **254**
- file select control **222**
- submission of **246**
- #FIXED attribute value **34**
- floating objects **197**
- floating text **198**
- focus **241**
- and access key **243**
- label gives to control **239**
- font
 - style with HTML **199**
- form
 - adding labels to **237**
 - content types for encoding **247**
 - control types **221**
 - controls in **220**
 - display notes **347**
 - encoding data of **246**
 - methods and actions **247**
 - navigating through controls **241**
 - processing controls of **246**
 - reset of **220**
 - structuring controls in **240**
 - submission method of **245**
 - submission of **245**

- tabbing order of controls **241**
- values submitted 245
- form data set **246**
 - encoding 246
- fragment identifier **20**, 146
- frame
 - URI problems with 213
 - border of 210
 - initial contents of 210
 - inline 217
 - introduction to 205
 - list of reserved target names 58
 - long description of 215
 - target algorithm 350
 - target of document 212
 - white space around 210
- frameset
 - DTD, declaration of **60**
 - DTD, definition of 297
 - alternate content for 214
 - navigation problems with 213
 - nested 208
 - sharing data among 208
 - specifying layout of 207
 - use of NOFRAMES in 215
- frameset document **206**
- %FrameTarget; **57**
- GET
 - and form submission 245
- header cell
 - abbreviation 136
 - in table 127
 - scope of 136
- headings
 - properly nested 76
- hidden control 222, 246
- HTML
 - as SGML application 39
 - authoring tips 24
 - comments in 30
 - development of 21
 - specifying data external to 336
 - version 2.0 21
 - version 3.0 21
 - version 3.2 21

- version HTML+ 21
- HTML document **37**
- HTML Working Group
 - members of 17
- HTTP
 - Content-Language header 90
 - Content-Script-Type header 253
 - Content-Style-Type header 186
 - Content-Type header 43
 - Default-Style header 191
 - GET and POST with forms 245
 - used to link external style sheets 194
 - hyphenation 96
- ID 50
- id attribute
 - roles of 71
 - same name space as name attribute 153
- IDREF 50
- IDREFS 50
- image
 - alignment of 180
 - border around 180
 - long description of 162
 - not directly in frame 215
 - visual rendering of 179
 - ways to include 160
 - white space around 180
 - width and height of 179
- image map 173, 179
 - accessibility of 176
 - client-side **174**
 - illegal for IMG in BUTTON 230
 - overlapping regions of 176
 - server side 179
 - server-side **174**
 - with OBJECT 176
- #IMPLIED attribute value **34**
 - including an object 164
 - inline
 - element **73**
 - %inline; 32
 - inter-word space **90**
 - Internet Engineering Task Force (IETF) 21
 - intrinsic events 254

- label
 - and focus 239
 - explicit association with control 237
 - implicit association with control 238
- lang attribute
 - not for direction 82
 - when applicable 79
- language
 - codes to specify 80
 - of linked resource 148
 - of script 253
 - of text 79
- %LanguageCode; **53**
- %Length; **52**
- line break **95**
 - and bidirectional text 96
 - and floating text 198
 - forcing 96
 - prohibiting 96
- link
 - and character encoding 148
 - and external style sheets 191, 155
 - and media-dependent style sheets 192
 - default target for 213
 - definition of **145**
 - forward and reverse 155
 - nesting illegal 152
 - rendering of 150
 - semantics with target frame 214
 - title of 148
 - type of 54
 - used to define relationship 147
 - used to retrieve resource 145
- link type
 - case of 54
 - list of recognized 54
 - profiles for new 56
- list
 - definition list **106**
 - nesting of 106
 - numbering of 106
 - ordered **104**
 - style sheets and 108
 - unordered **104**
 - visual rendering of 108

- long image description
 - relation to alt text 162
- markup **27**
- markup language **27**
- media
 - and external style sheets 192
 - used with style sheets 189
- media descriptor
 - case of 57
 - list of recognized 56
 - parsing of 56
- %MediaDesc; **56**
- menu 222
 - grouping of choices 231
 - preselected options 231
 - rendering of choices 233
 - visual rendering of grouped options 234
- message entity 39
- meta data **64**
 - LINK vs META 66
 - profiles for **68**
 - scheme for 68
- %MultLength; **52**
- multipart/form-data **248**, 223
- NAME 50
- notes about minimized 337
- NUMBER 50
- numbered headings
 - numbered 76
- numeric character reference **45**
- object
 - alignment of 180
 - border around 180
 - fallback rendering of 165
 - generic inclusion 164
 - in HEAD 208, 165, 165
 - in form 222
 - initialization 167
 - locating implementation and data 165
 - naming schemes for 169
 - rules for embedded 165
 - statically declared 169
 - visual rendering of 179
 - white space around 180

- width and height of **179**
- object control **222**, **246**
- obsolete **38**
- elements **328**
- paragraph
 - visual rendering of **98**
- parameter entity
 - %Character; **53**
 - %Charset; **53**
 - %Color; **51**
 - %ContentType; **53**
 - %Datetime; **54**
 - %FrameTarget; **57**
 - %LanguageCode; **53**
 - %Length; **52**
 - %MediaDesc; **56**
 - %MultiLength; **52**
 - %Pixels; **52**
 - %Script; **57**
 - %Text; **50**
 - %URI; **51**
 - %attrs; **35**
 - %block; **32**
 - %inline; **32**
- parameter entity definition **31**
- password input control **226**
- persistent style sheets **190**
- pixel **52**
- %Pixels; **52**
- Platform for Internet Content Selection (PICS) **67**
- POST
 - and form submission **245**
 - for non-ASCII form data **245**
- preferred style sheets **190**
- profile **68**
- push button **221**
- quoted text **93**
- rendering of **93**
- radio button **221**
- read only controls **244**
- relative length **52**
- relative URI **20**
- resolution of **20**

- #REQUIRED attribute value **34**
- reset button **221**
- resetting a form **220**
- resolution of relative URI **158**
- Resource Description Framework (RDF) **24**, **64**
- row
 - number of in table **114**
- row group **117**
- rule
 - between block-level elements **202**
 - between table cells **130**
- scheme **68**
- scope
 - of table header cell **136**
- script
 - comments to hide **259**
 - data **57**
 - executed on event **251**
 - executed when document loaded **251**
 - implementation notes **348**
 - introduction to **251**
 - references to elements **254**
 - reserved syntax for **348**
 - used to modify document **258**
 - used to set anchor **150**
 - uses of **251**
 - when unsupported **258**
- %Script; **57**
- scripting language
 - default **253**
 - local declaration **254**
 - specification of **253**
- search engine
 - and links **155**
 - helping **339**, **66**
- search robot
 - helping **340**
- security
 - notes on **350**
 - of password control **226**
- server-side image map **174**, **179**
- click coordinates **179**
- SGML
 - application **28**
 - catalog for HTML **262**
 - declaration **28**

- declaration of HTML 4 **263**
- document character set **41**
- document type definition (DTD) **28**
- document validation 261
- element type declaration **28**
- features with limited support 337, 337
- implementation notes 335
- introduction to 27
- treatment of line breaks 335
- soft hyphen **96**
- start tag **28**
- strict DTD
 - omitted 28
 - declaration of **60**
 - definition of 265
 - style sheet
 - alternate **190**
 - and bidirection 88
 - cascading 192
 - comments to hide 193
 - data 57
 - external 190
 - external through links 155
 - inline rules 186
 - introduction to 183
 - persistent **190**
 - preferred **190**
 - rules in HEAD 187
 - specification of external 191
 - specification of preferred 191
 - target media for 189
 - used with DIV and SPAN 188
 - style sheet language
 - default 186
 - submit button 221
 - successful control 245
 - summary
 - of table contents 116
- tabbing navigation 241
- tabbing order **241**
- table
 - algorithm to find heading 142
 - alignment of contents 132
 - borders and rules of 130
 - caption for 116
 - categorization of cells 139

- cell margins 134
- cells that span several rows/columns 128
- column group in 118
- data cells 127
- directionality of 115
- header cells 127
- incremental display notes 342
- incremental rendering of 114
- layout algorithms for 344
- non-visual rendering of 136
- not for formatting pages 23
- number of columns 121
- number of rows 114
- row group in 117
- speaking cell data 142
- summary of contents 116
- visual rendering of 130
- width of columns 122
- target frame
 - algorithm 350
 - default 213
 - reserved names 58
 - semantics of 214
 - specification of 212
- text
 - direction of 82
 - floating 198
 - markup for inserted and deleted 100
 - preformatted 97
 - quoted 93
 - wrapping in paragraph 99
- text input control 222
 - multi-line 235
 - single-line 226
- text/html 39
- %Text: **50**
- time
 - format of 54
- title
 - available to user 63
 - of a document 63
 - used to annotate elements 63
- transitional DTD
 - declaration of **60**
 - definition of 279

- Unicode bidirectional algorithm 82
- universal character set 41
- Universal Resource Identifier (see URI) **19**
- URI
 - case of **51**
 - non-ASCII characters in attribute values 334
 - relative **20**
 - resolution of relative **20**, 158
 - specifying base 157
 - uses of in HTML 21
 - %URI; **51**
- URL
 - relationship to URI **20**
- user agent **37**
- user agent **37**
 - and error conditions **334**, **38**
 - and script data 259
 - and style data 193
 - conforming 38
 - handling image maps 176
 - processing script and style data 50
- UTF-1 43
- UTF-16 43
- white space 89
 - around frame 210
 - around images and objects 180
 - around table contents 134
- character **89**
- collapsed 90
- preserved in PRE 97
- World Wide Web (Web) **19**
- wrapping text 99