HTML 4.0 Specification

W3C Working Draft *8-July-1997*

This is: http://www.w3.org/TR/WD-html40-970708/

Abstract

This specification defines the HyperText Markup Language (HTML), version 4.0, the publishing language of the World Wide Web. In addition to the text, multimedia, and hyperlink features of the previous versions of HTML, HTML 4.0 supports more multimedia options, scripting languages, style sheets, better printing facilities, and documents that are more accessible to users with disabilities. HTML 4.0 also takes great strides towards the internationalization of documents, with the goal of making the Web truly World Wide.

Status of this document

This is a W3C Working Draft for review by W3C members and other interested parties. It is a draft document and may be updated, replaced or obsoleted by other documents at any time. It is inappropriate to use W3C Working Drafts as reference material or to cite them as other than "work in progress". This is work in progress and does not imply endorsement by, or the consensus of, either W3C or members of the HTML working group.

This document has been produced as part of the W3C HTML Activity, and is intended as a draft of a proposed recommendation for HTML.

The latest version of this document can be retrieved from the list of W3C technical reports at <http://www.w3.org/TR/> and is available as a gzip'ed tar file, a zip file, as well as a postscript (about 200 pages).

We also plan to provide translations in other languages, although the English version provides the normative specification.

HTML 4.0 replaces HTML 3.2, specified in http://www.w3.org/TR/REC-html32.

Editors

- Dave Raggett <dsr@w3.org>
- Arnaud Le Hors <lehors@w3.org>
- Ian Jacobs <ij@w3.org>

Comments

Please send detailed comments on this document to www-html-editor@w3.org. We cannot garantee a personal response but we will try when it is appropriate. Public discussion on HTML features takes place on www-html@w3.org.

Table of Contents

About the HTML 4.0 Specification

Contents

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

The specification has been written with two modes of presentation in mind: electronic and printed. Although the two presentations will no doubt be similar, readers will find some differences. For example, links will not work in the printed version (obviously), and page numbers will not appear in the electronic version. In case of a discrepancy, the electronic version is considered the authoritative version of the document.

How to read the specification

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. This is reflected in the specification's main table of contents, which presents topical information, and the indexes, which present lower level information in alphabetical order.

- **Quick access to information.** In order to get information about syntax and semantics as quickly as possible, the electronic version of the specification includes the following features:
  1. Every reference to an element or attribute is linked to its definition in the specification.
  2. Every page will include links to the indexes, so you will never be more than two links away from finding the definition of an element or attribute.
  3. The front pages of the three sections of the language reference manual extend the initial table of contents with more detail about each section.

How the specification is organized

This specification includes the following sections:

Section 2: Introduction to HTML 4.0.
The introduction gives an overview of what can be done with HTML 4.0. It also provides some design tips for developing good HTML habits.

Sections 3 - 11: HTML 4.0 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 designers' experience that separating a document's structure from its presentation produces more effective and maintainable documents.

The language reference consists of the following information:

- Conventions used by the editors of this specification.
- How HTML fits into the World Wide Web and an introduction to related Web languages and protocols such as URLs.
- What characters may appear in an HTML document.
- Basic data types of an HTML document.
- Elements that pertain to the structure of an HTML document, including text, lists, tables, links, and included objects, images, and applets.
- Elements that pertain to the presentation of an HTML document, including style sheets, fonts, colors, rules, and other visual presentation, and frames for multi-windowed presentations.
- Elements that pertain to interactivity with an HTML document, including forms for user input and scripts for active documents.
- The SGML definition of HTML, including the SGML declaration of HTML, the HTML DTD, and the list of character entities.
- References.

Section 12: Quick reference indexes.

Two indexes give readers rapid access to the definition of all elements and attributes The indexes also summarize some key characteristics of each element and attribute.

Section 13: Appendixes.

The appendix contains information about changes from HTML 3.2, performance and implementation notes, and how W3C and other organizations interact with respect to HTML.

Acknowledgments

Thanks to everyone who has helped to author the working drafts that went into the HTML 4.0 specification, and all those who have sent suggestions and corrections. A particular thanks to T.V. Raman for his work on improving the accessibility of HTML forms for people with disabilities.

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, Steve Byrne, Martin J. Dürst, Daniel Glazman, Scott Isaacs, Murray Maloney, Steven Pemberton, Jared Sorensen, Powell Smith, Robert Stevahn, Ed Tecot, Jeffrey Veen, Mike Wexler, Misha Wolf, and Lauren Wood.

Thank you Dan Connolly for thoughtful input and guidance as chairman of the HTML working group. Thank you Sally Khudairi for your indispensible work on the press release.

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

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

Introduction to HTML 4.0

Contents

This is being written ...

Design principles of HTML 4.0

As you read the specification, you may find it enlightening to keep in mind the following principles that guided the design of HTML 4.0.

- **Interoperability**

  While most people agree that HTML documents should work well across different browsers and platforms, achieving interoperability implies higher costs to content providers since they must develop different versions of documents. If the effort is not made, however, 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 attempts to reach 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.

- **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], which deals with the internationalization of HTML.

  One important step has been the adoption of the ISO/IEC:10646 standard (see [ISO10646]) 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, correct hyphening, etc.

- **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 4.0 developments in the area of accessibility include:

  - Encouraging the use of style sheets (rather than tables) to achieve layout effect.
  - Making it easier to provided alternate (textual and aural) descriptions of images for non-visual browsers.

- Providing active labels for form fields
- Providing labeled hierarchical groupings for form fields.
- Providing the ability to associate a longer text description with an HTML element.

Authors who design pages with accessibility issues in mind will not only receive the blessings of the accessbility 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.

- **Tables**

  The new table model in HTML is based on [RFC1942]. 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.

- **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.

- **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.

  Stylistic information can be:

  - Attached to a specific element to affect, say the color or font of its content.
  - Placed in the document header as a series of styles comprising a style sheet
  - Linked to an HTML from an external style sheet.

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

- **Scripting**

  Through scripts, authors may create "smart forms" that react as users fill them out. Scripting allows designers to create dynamic Web pages, and to use HTML as a means to build networked applications. The mechanisms provided to associate HTML with scripts are independent of particular scripting languages.

- **Printing**

HTML features allow user agents to print a collection of documents in an intelligent manner based on descriptions of the relationships among documents acting as parts of a larger work.

- **Ease of use**

  This version of HTML has been designed to remain easy to learn and adequate for many common publishing needs. The language offers more complex constructs (e.g., forms, scripting) for more sophisticated tasks, but even these mechanisms will become easier to use as powerful HTML authoring tools flourish.

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

Designing documents with HTML 4.0

General principles for good HTML design and implementation include:

- **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.

- **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 designers 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, designers should include in their documents information about the language and direction of the text, how the document is encoded, and other issues related to internationalization.

- **Help user agents with incremental rendering**

  By carefully designing their tables and making use of new table features in HTML 4.0, designers can help user agents render documents more quickly.

A brief SGML tutorial

Contents

This section of the document presents introductory information about SGML and its relationship to HTML. It discusses:

- HTML syntax: How to write elements, attributes, and comments.
- The HTML DTD: How to read the HTML DTD.

About SGML

The *Standard Generalized Markup Language* (SGML, defined in [ISO8879]), is a language for defining markup languages. HTML is one such "application" of SGML.

An SGML application consists of several parts:

1. The SGML declaration. The SGML declaration specifies which characters and delimiters may appear in the application.
2. The document type definition (DTD). The DTD defines the syntax of markup constructs. The DTD may include additional definitions such as numeric and named character entities.
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 (contents) and markup. Each instance contains a reference to the DTD to be used to interpret it.

The SGML declaration for HTML 4.0 and the DTD for HTML 4.0 are included in this reference manual, along with the entity sets referenced by the DTD.

HTML syntax

In this section, we discuss the syntax of HTML elements, attributes, and comments.

Entities

Character entities are numeric or symbolic names for characters that may be included in an HTML document. They are useful when your authoring tools make it difficult or impossible to enter a character you may not enter often. You will see character entities throughout this document; they begin with a "&" sign and end with a semi-colon (;).

We discuss HTML character entities in detail later in the section on the HTML document character set.

Elements

An SGML application defines elements that represent structures or desired behavior. An *element* typically consists of three parts: a start tag, content, and an end tag.

A element's start tag is written `<element-name>`, where `element-name` is the name of the element. An element's end tag is written with a slash before the element name: `</element-name>`. For example,

```
<pre>The content of the PRE element is preformatted text.</pre>
```

The SGML definition of HTML specifies that some HTML elements are not required to have end tags. The definition of each element in the reference manual indicates whether it requires an end tag.

Some HTML elements 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 definition of each element in the reference manual indicates whether it is empty (has no content) or, if it can have content, what is considered legal content.

Element names are always case-insensitive.

Elements are not tags. *Some people refer incorrectly 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.*

Attributes

Elements may have associated properties, called *attributes*, to which authors assign values. 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 `align` attribute is set for the `H1` element:

```
<H1 align="center">
This is a centered heading thanks to the align attribute
</H1>
```

By default, SGML requires you to delimit all attribute values using either double quotation marks (") or single quotation marks ('). Single quote marks can be included within the attribute value when the value is delimited by double quote marks, and vice versa. You may also use numeric character entities to represent double quotes (") and single quotes ('). For double quotes you can also use the named character entity ".

In certain cases, it is possible in HTML to 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), and periods (ASCII decimal 46). We suggest 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.

Note: HTML documents may compress better if you use lower case letters for element and attribute names. The reason is that the compression algorithms do a better job for more frequently repeated patterns, and lower case letters are more frequent than upper case ones.

HTML comments

HTML comments have the following syntax:

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

Comments must not be rendered by user agents as part of a document. Similary, user agents must not render SGML processing instructions (e.g., <?full volume>).

How to read the HTML DTD

This specification presents pertinent fragments of the DTD each time an element or attribute is defined. Though cryptic and dissuasive at first, the DTD fragment gives concise information about an element and its attributes. 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. While almost all of the definitions include enough English text to make them comprehensible, for those who require definitive information, we complete this specification with a brief tutorial on reading the HTML DTD.

Block level and Inline elements

Certain HTML elements 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 generally 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 are formatted differently than inline elements. Block level elements generally begin on new lines, inline elements generally do not. Block level elements end an unterminated paragraph element. This enables you to omit end-tags for paragraphs in many cases.

Directionality
: For technical reasons involving the [UNICODE] 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.

Style sheets 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 for more information.

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. DTD comments for HTML do have not normative value.

Entity Definitions

The HTML DTD begins with a series of entity definitions. An *entity definition* (not to be confused with an SGML entity) defines a kind of macro that may be expanded elsewhere in the DTD. When the macro is referred to by name in the DTD, it is expanded into a string.

An entity definition begins with the keyword `<!ENTITY %` followed by the entity name, the quoted string the entity expands to, and finally a closing >. The following example defines the string that the `%font` entity will expand to.

```
<!ENTITY % font "TT | I | B | U | S | BIG | SMALL">
```

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

```
<!ENTITY % inline "#PCDATA | %font | %phrase | %special | %formctrl">
```

You will encounter two DTD entities frequently in the HTML DTD: %inline and %block. They are used when the content model includes inline and block level elements respectively.

Element definitions

The bulk of the HTML DTD consists of the definitions of elements and their attributes. The <!ELEMENT> keyword begins an element definition and the > character ends it. Between these are specified:

1. The element's name.
2. Whether the element's end tag is 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" (not zero) 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*. Elements with no content are called *empty* elements. Empty elements are defined with the keyword "EMPTY".

In this example:

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

- The element being defined is **UL**.
- The two hyphens indicate that both the start tag and the end tag for this element are required.
- The content model for this element defined to be "at least one LI element". We describe content models in detail below.

This example illustrates the definition of an empty element:

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

- The element being defined 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 the element must not have content.

Content model definitions

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

- The names of allowed or forbidden elements (e.g., the `UL` element includes instances of the `LI` element).
- DTD entities (e.g., the `LABEL` element includes instances of the %inline entity).
- Document text (indicated by the SGML construct "#PCDATA"). Text may contain numeric and named character entities. Recall that these begin with & and end with a semicolon (e.g., "Hergé's adventures of Tintin" includes the named entity for the "acute e" character).

The content model use the following syntax to define what markup is allowed for the content of the element:

(...)
: Specifies a group.

A | B
: Both A and B are permitted in any order.

A , B
: A must occur before B.

A & B
: A and B must both occur once, but may do so in any order.

A?
: A can occur zero or one times

A*
: A can occur zero or more times

A+
: A can occur one or more times

Here are some examples from the HTML DTD:

```
<!ELEMENT SELECT - - (OPTION+)>
```

The `SELECT` element must contain one or more `OPTION` 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 &

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

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

```
<!ELEMENT A - - (%text)* -(A)>
```

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

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

```
<!ELEMENT FORM - - %block -(FORM)>
```

Attribute definitions

The <!ATTLIST> keyword begins the definition of attributes that an element may take. It is followed by the name of the element in question and a list of attribute definitions. An 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.
- 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 attributes 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 include character entities.

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

The following examples illustrate possible 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 include DTD entities.

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

```
<!ATTLIST LINK
  %attrs;                          -- id, class, style, lang, dir, title --
  href        %URL       #IMPLIED  -- URL for linked resource --
  ...more of the definition...
  >
```

The %attrs entity expands to:

```
<!ATTLIST P
  id          ID         #IMPLIED  -- document-wide unique id --
  class       CDATA      #IMPLIED  -- comma list of class values --
  style       CDATA      #IMPLIED  -- associated style info --
  title       CDATA      #IMPLIED  -- advisory title/amplification --
  lang        NAME       #IMPLIED  -- [RFC1766] language value --
  dir         (ltr|rtl)  #IMPLIED  -- direction for weak/neutral text --
  align (left|center|right|justify)  #IMPLIED
  >
```

The %attrs entity has been defined for convenience since these seven attributes are defined for most HTML elements.

Simiarly, the DTD defines the %URL entity as expanding into the string CDATA.

```
<!ENTITY % URL "CDATA"
        -- The term URL means a CDATA attribute
           whose value is a Uniform Resource Locator,
           See [RFC1808] and [RFC1738]
        -->
```

As this example illustrates, the entity %URL 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, %Content-Type, %Length, %Pixels, etc.

Boolean attributes

Some attributes play the role of boolean variables (e.g., `selected`). 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  -- reduced interitem spacing --
```

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

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

Minimized boolean attributes *In HTML, boolean attributes may be appear in "minimized form" -- the attribute’s* ***value*** *appears alone in the element’s start tag. Thus:*

```
<OPTION selected>
```

instead of

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

Authors should be aware than many user agents only recognize the minimized form and not the full form.

Definitions and Conventions

Contents

Below we list some definitions and conventions adopted in this specification.

Words such as "must", "should", "can", and "may" are used in accordance with [RFC2119].

Definitions

User agent A user agent is any device that interprets HTML documents. User agents include visual browsers (ascii and graphical), non-visual browsers (audio, braille), search robots, proxies, etc.

Conforming user agent A conforming user agent for HTML 4.0 is one that observes the mandatory conditions set forth in this specification.

A user agent must try to render the content of any element it does not recognize.

A user agent must ignore any attribute it does not recognize.

A user agent should avoid imposing arbitrary length limits on string literals.

This specification does not define how conforming user agents should handle general error conditions.

Deprecated A deprecated element or attribute is one that has been outdated by newer HTML 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.

We strongly urge authors to avoid deprecated elements and attributes. To this end, we provide alternatives to them when appropriate in the specification.

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

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 purpose in the changes section of the reference manual.

Document conventions

This specification presents elements in a "semantic" order, from most basic to most sophisticated constructs. Elements may be discussed in several different contexts depending on their use (e.g., `DIV` as a structuring element and `DIV` and its role with style sheets). Despite scattered references, the specification defines each element and attribute in one location only.

Elements and attributes

Similarly, attributes that apply to many elements (e.g., `lang`, `dir`, `class`, etc.) are defined where most appropriate semantically. Consequently, an element definition may include a reference to an attribute that may be defined elsewhere. When this is the case, the location of the definition will be clearly indicated and for the electronic version of the specification, accessible by a link.

Definitions of elements and attributes clearly indicate which are deprecated. In such cases, the specification includes examples of better HTML usage.

In the electronic version of the specification, all references to an element or attribute (from the index or in the specification text) are linked to its definition.

Element names are written in upper case letters (e.g., BODY). Attribute names are written in lower case letters (e.g., lang, onsubmit). Recall that in HTML, element and attribute names are case-insensitive. Our convention is designed to encourage readability of the specification.

Element and attribute names are marked within the source HTML for the specification and may be rendered specially depending on your user agent.

The type of an attribute's value is specified in its definition. However, if the set of possible values is small, the values are listed explicitly, separated by a bar (|). The first value in this list is the default value.

Notes and examples

Informative notes will be emphasized to stand out from surrounding text. How the emphasis is rendered depends on your user agent.

An example that illustrates deprecated usage will be marked as a "DEPRECATED EXAMPLE". Deprecated examples also include recommended alternate solutions. An example that illustrates illegal usage will be clearly marked as an "ILLEGAL EXAMPLE".

Examples and notes are marked within the source HTML for the specification and may be rendered specially depending on your user agent.

Document names

By convention, HTML files are usually given the extension ".html" or ".htm".

SGML

Comments appearing in the HTML 4.0 DTD have no normative value; they are informative only.

HTML and URLs

Contents

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

1. A single naming scheme, to give access to any resource on the Web in a uniform way (URLs).
2. Protocols, to enable the exchange of named resources over the Web (HTTP).
3. Hypertext, for easy navigation among resources (HTML).

In this section of the reference manual, we present (minimal) information about those Web topics that have an impact on HTML.

Universal Resource Locators (URLs)

Every resource available on the Web --- HTML document, image, video clip, program, etc. --- has an address that may be encoded by a *Universal Resource Locator*, or "URL" (defined in [RFC1738]).

URLs typically consist of three pieces:

1. The name of the protocol used to transfer the resource over the Web.
2. The name of the machine hosting the resource.
3. The name of the resource itself, given as a path.

Consider the URL that designates the current HTML specification:

```
http://www.w3.org/TR/WD-html4/cover.html
```

This URL may be read as follows: Use the HTTP protocol to transfer the data residing on the machine www.w3.org in the file /TR/WD-html4/cover.html

URLs in general are case-sensitive (with the exception of machine names). There may be URLs, or parts of URLs, where case doesn't matter, but identifying these may not be easy. Users should always consider that URLs are case-sensitive. /u

The character set of URLs that appear in HTML is specified in [RFC1738].

Fragment URLs

The URL specification *en vigeur* at the writing of this document ([RFC1738]) offers a mechanism to refer to a resource, but not to a location within a resource. The Web community has adopted a convention called "fragment URLs" to refer to anchors within an HTML document. A fragment URL ends with "#" followed by an anchor identifier. For instance, here is a fragment URL pointing to an anchor named `section_2`:

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

Relative URLs

A *relative* URL (defined in [RFC1808]) doesn't contain any protocol or machine information, and its path generally refers to an HTML document on the same machine as the current document. Relative URLs may contain relative path components (".." means the parent location) and may be fragment URLs.

Relative URLs may be resolved to full URLs, for example when the user attempts to follow a link from one document to another. [RFC1808] defines the normative algorithm for resolving relative URLs. The following description is for convenience only.

Briefly, a full URL is derived from a relative URL by attaching a "base" part to the relative URL. The base part is a URL that may come from any or all of the following sources:

- HTTP transfer protocol header information (see [RFC2068]).
- Metadata (the `META` element) in the `HEAD` section of an HTML document.
- Explicit base path information (the `BASE` element) in the `HEAD` section of an HTML document, or the `CODEBASE` attribute of the `APPLET` element.

[RFC1808] specifies the precedence among multiple sources of base information. For the purposes of this explanation, the last piece of base information takes precedence over the others and HTTP headers are considered to occur earlier than the document HEAD.

If no explicit base information accompanies the document, the base URL is that which designates the location of the current document.

Given a base URL and a relative URL (that does not begin with a slash), a full URL is derived as follows:

- If the base URL ends with a slash the full URL is derived by appending the relative URL to the base URL. For example, if the base URL is `http://nosite.com/dir1/dir2/` and the relative URL is `gee.html`, the derived URL is `http://nosite.com/dir1/dir2/gee.html`.
- If the base URL doesn't end with a slash, the last piece of the base URL is considered a resource, so the full URL is derived by appending the relative URL to the parent of the base URL. For example, if the base URL is `http://nosite.com/dir1/dir2` and the relative URL is `gee.html`, the derived URL is `http://nosite.com/dir1/gee.html`

URLs in HTML

In HTML, URLs play a role in these situations:

- When referring to metadata describing a document (see the `HEAD` element).
- When citing a external reference (see the `Q`, `BLOCKQUOTE`, `INS`, and `DEL` elements).
- When including an object directly in a document (see the `OBJECT`, `IMG`, `MAP`, `FRAME`, and `IFRAME` elements).
- When linking to another document or program (see the `BASE`, `A`, `LINK`, `AREA`, `FORM`, `INPUT`, `SCRIPT`, and `APPLET` elements).

In each case, authors may use a full URL, a fragment URL, or a relative URL. Please consult the section on anchors for more information about links and URLs.

MAILTO URLs

In addition to HTTP URLs, authors might want to include MAILTO URLs (see [RFC1738]) in their documents. MAILTO URLs cause email to be sent to some email address. For instance, the author might create a link that, when activated, causes the user agent to open a mail program with the destination address in the "To:" field.

MAILTO URLs have the following syntax:

```
mailto:email-address
```

User agents may support MAILTO URL extensions that are not yet Internet standards (e.g., appending subject information to a URL with the syntax "?Subject=my%20subject" where any space characters are replaced by "%20").

HTML Document Character Set

Contents

Human languages define a large number of text characters and human beings have invented a wide variety of systems for representing these characters in a computer. Unless proper precautions are taken, differing character representations may not be understood by user agents in all parts of the world.

The Document Character Set

To promote interoperability, SGML requires that each application (including HTML), as part of its definition, define its *document character set*. A document character set is a set of abstract characters (such as the Cyrillic letter "I", the Chinese character meaning "water", etc.) and a corresponding set of integer references to those characters. SGML considers a document to be a sequence of references in the document character set.

The document character set for HTML is the Universal Character Set (UCS) of [ISO10646]. This set is character-by-character equivalent to Unicode 2.0 ([UNICODE]). 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, references to ISO/IEC-10646 or Unicode imply the same document character set. However, the current document also refers to the Unicode specification for other issues such as the bidirectional text algorithm.

Conforming HTML user agents may receive or output a document, or represent a document internally, using any *character encoding*. A character encoding represents some subset of the document character set. Character encodings such as ISO-8859-1 (commonly referred to as "Latin-1" since it encodes most Western European languages), ISO-8859-5 (which supports Cyrillic), SHIFT_JIS (a Japanese encoding), and euc-jp (another Japanese encoding) save bandwidth by representing only slices of the document character set.

Thus, character encodings allow authors to work with a convenient subset of the document character. Authors should not have to know anything about the underlying character encoding of the document or tool they are using --- writing Japanese in a UTF-8 editor is as easy as writing Japanese in a JIS or SHIFT_JIS editor.

Character encodings also mean that authors are not required to enter a document's text in the form of references the document character set. Requiring authors to work with such a large character encoding would be cumbersome and wasteful (although encodings such as UTF-8 that cover all of Unicode do exist).

To allow this convenience, conforming user agents must correctly map to [UNICODE] all characters in any character encodings ("charsets") they recognize (or behave as if they did). A list of recommended character encodings for various scripts and languages will be provided in a separate document.

How does a user agent know which character encoding has been used to encode a given document?

In many cases, before a Web server sends an HTML document over the Web, it tries to figure out the character encoding (by a variety of techniques such as examining the first few bytes of the file, checking its encoding against a database of known files and encodings, etc.). The server transmits the document and the name of the character encoding to the receiving user agent by way of the `charset` parameter of the HTTP "Content-Type" field. For example, the following HTTP header announces that the character encoding is "euc-jp".

```
Content-Type: text/html; charset=euc-jp
```

The value of the "charset" parameter must be the name of a "charset" as defined in [RFC2045].

Unfortunately, not all servers send information about the character encoding (even when the character encoding is different from the widely used ISO-8859-1 encoding). HTML therefore allows authors a way to tell user agents which character encoding has been used by specifying it explicitly in the document header with the `META` element. For example, to specify that the character encoding of the current document is "euc-jp", include the following `META` declaration:

```
<META http-equiv="Content-Type" Content="text/html; charset=euc-jp">
```

This mechanism has a notable limit: the user agent cannot interpret the `META` element to determine the character encoding if it doesn't already know the character encoding of the document. The `META` declaration must only be used when the character encoding is organized such that ASCII characters stand for themselves at least until the `META` element is parsed. In this case, conforming user agents must correctly interpret the `META` element.

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

1. Explicit user action to override erroneous behavior.
2. An HTTP "charset" parameter in a "Content-Type" field.
3. A `META` declaration with "http-equiv" set to "Content-Type" and a value set for "charset".
4. The "charset" attribute set for the `A` and `LINK` elements.
5. User agent heuristics and user settings. For example, user agents typically assume that in the absence of other indicators, the character encoding is **ISO-8859-1**. This assumption may lead to an unreadable presentation of certain documents.

In all cases, the value of the "charset" attribute or parameter must be the name of a "charset" as defined in [RFC2045].

If, for a specific application, it becomes necessary to refer to characters outside [ISO10646], 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.

Note: Modern web servers can be configured with information about which document is using which character encoding. Webmasters should use these facilities but should take pains to configure the server properly.

Character entities

Your hardware and software configuration probably won't allow you to refer to all Unicode characters through simple input mechanisms, so SGML offers character encoding-independent mechanisms for specifying any character from the document character set.

- Numeric character references (either decimal or hexadecimal form).
- Named character references.

Numeric character references specify the integer reference of a Unicode character. A numeric character reference with the syntax &#D; refers to Unicode decimal character number D. A numeric character reference with the syntax &#xH; refers to Unicode hexadecimal character number H. The hexadecimal representation is a new SGML convention and is particularly useful since character standards use hexadecimal representations.

Here are some examples:

- Entity å refers to the letter "a" with a small circle above it (used, for example, in Norwegian).
- Entity å refers to the same character with the hexadecimal representation.
- Entity И refers to the Cyrillic capital letter "I".
- Entity 水 refers to the Chinese character for water with the hexadecimal representation.

To give authors a more intuitive way to refer to characters in the document character set, HTML offers a set of *named character entities*. Named character references replace integer references with symbolic names. The named entity å refers to the same Unicode character as å. There is no named entity for the Cyrillic capital letter "I". The full list of named character entities is included in this specification.

Four named character entities deserve special mention since they are frequently used to "escape" special characters: For text appearing as part of the content of an element, you should escape < as < to avoid possible confusion with the beginning of a tag. The & character should be escaped as & to avoid confusion with the beginning of an entity reference.

You should also escape & within attribute values since entity references are allowed within cdata attribute values. In addition, you should escape > as > to avoid problems with older user agents that incorrectly perceive this as the end of a tag when coming across this character in quoted attribute values.

Rather than worry about rules for quoting attribute values, its often easier to encode any instance of " by " and to always use " for quoting attribute values. Many people find it simpler to always escape these 4 characters in element content and attribute values.

- "&" to represent the & sign.
- "<" to represent the < sign.
- ">" to represent the > sign.
- "" to represent the " mark.

Names of named character entities are case-sensitive. Thus, Å refers to a different character (upper case A, ring) than å (lower case a, ring).

Note: In SGML, it is possible to eliminate the final ";" after a numeric or named character reference in some cases (e.g., at a line break or directly 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.

Basic HTML data types

Contents

URLs

The type "url" refers to either an absolute or relative Universal Resource Locator. Please consult the section on URLs for more details.

Character data

The syntax of valid character data in HTML is defined in terms of the SGML concepts of NAME and CDATA. For more introductory information about SGML, please consult the SGML tutorial. For more information about SGML, please consult the SGML handbook ([GOLD90]).

- *NAME*. SGML name tokens must begin with a letter (A-Z and a-z) and may be followed by any number of letters, digits, hyphens ("-") and periods (".").
- *CDATA*. For attribute values, this refers to a sequence of characters from the document character set, which may include character entities. User agents should resolve CDATA before further processing of attribute values. The HTML 4.0 specification often places further constraints on the permitted syntax for CDATA attributes.

  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 "</" is treated as terminating the end of the element's content. In valid documents, this would be the end tag for the element.

Colors

The attribute value type "color" refers to color definitions as specified in [SRGB]. A color value may either be a hexadecimal number (prefixed by a hash mark) or one of the following sixteen color names:

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.

Notes on using colors

Although colors can add significant amounts of information to document 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. You are encouraged to use style sheets 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] color model together with ICC color profiles should mitigate this problem.
- When practical, adopt common conventions to minimize user confusion. Unless, of course, your desired effect is to confuse users!

Lengths and Pixels

Values of the type "length" may either be specified as an integer representing the number of pixels of the canvas (screen, paper) or as a percentage of the available horizontal or vertical space. The HTML DTD generally uses %Length for length values that permit percentages and %Pixels for values that only permit pixels.

Thus, the value "50" means fifty pixels. For widths, the value "50%" means half of the available horizontal space (between margins, within a table cell, etc.). For heights, the value "50%" means half of the available vertical space (in the current window, the current table cell, etc.).

For normative information about the definition of a pixel, please consult [CSS1].

Structure of HTML documents

Contents

Global structure

Contents

An HTML 4.0 document generally consists of three parts: a line containing version information, a descriptive header section, and a body, which contains the document's actual content.

HTML version information

The SGML DOCTYPE construct declares which version of HTML was used in composing the document (see [GOLD90]).

Authors should include a declaration resembling the following as the first line of each document:

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.0 Draft//EN">
```

The last two letters of the declaration indicate the language of the HTML DTD, in this case English ("EN"). User agents may ignore this information.

Authors may employ a different document type description depending on the version of HTML their document relies on. Recommended document types for HTML 4.0 are:

<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.0 Draft//EN">

For documents following the draft specification for HTML 4.0

<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.0 Final//EN"> or <!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.0//EN">

For documents following the final specification for HTML 4.0

<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.0 Strict//EN">

For documents following the strict specification for HTML 4.0. This is appropriate when you wish to validate documents that don't use the HTML presentation elements and attributes, such as the FONT element and the align attribute.

```
<!DOCTYPE HTML SYSTEM "http://www.w3.org/MarkUp/Cougar/relaxed.dtd">
```

For validatation against the DTD on the W3C Web site

```
<!DOCTYPE HTML SYSTEM "http://www.w3.org/MarkUp/Cougar/strict.dtd">
```

For validation against the strict DTD on the W3C Web site

The binding between public identifiers and files can be specified using a catalog file following the format recommended by the SGML Open Consortium. A sample catalog file for HTML 4.0 is included at the beginning of the section on SGML reference information for HTML. *Note: Some user-agents do not understand more complicated DOCTYPE declarations than those listed above.*

The `HTML` element

```
<!ENTITY % version "version CDATA #FIXED '%HTML.Version;'">

<!ENTITY % html.content "HEAD, (FRAMESET|BODY)">

<!ELEMENT HTML O O (%html.content)>
<!ATTLIST HTML
  %version;
  %i18n;                          -- lang, dir --
  >
```

*Start tag: **optional**, End tag: **optional***

Attribute definitions

`version` = *url*
: This attribute specifies (with a URL) the location of the DTD for the version of HTML governing the current document. Since the same information must appear in the `DOCTYPE` header, the usefulness of this attribute is uncertain.

Attributes defined elsewhere

- `lang` (language information), `dir` (text direction)

After version information, the remainder of an HTML document should be enclosed by the `HTML` element. Thus, a typical HTML document has this structure:

```
<!DOCTYPE HTML PUBLIC -//W3C//DTD HTML 4.0 Draft//EN>
<HTML>
...The head, body, etc. goes here...
</HTML>
```

The HEAD element

```
<!-- %head.misc defined earlier on as "SCRIPT | STYLE | META | LINK" -->
<!ENTITY % head.content "TITLE & ISINDEX? & BASE?">

<!ELEMENT HEAD O O  (%head.content) +(%head.misc)>
<!ATTLIST HEAD
  %i18n;                           -- lang, dir --
  profile     %URL       #IMPLIED  -- named dictionary of meta info --
  >
```

*Start tag: **optional**, End tag: **optional***

Attribute definitions

`profile` = *url*
: 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 URL to be significant. Profiles are discussed below in the section on meta information.

Attributes defined elsewhere

- `lang` (language information), `dir` (text direction)

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. Elements within the `HEAD` declaration must not be rendered by conforming user agents unless otherwise specified.

Titles: the TITLE element and the title attribute

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

*Start tag: **required**, End tag: **required***

Attributes defined elsewhere

- `lang` (language information), `dir` (text direction)

Every HTML document **must** have exactly one `TITLE` element in the `HEAD` section. User agents generally use the title to give people some idea about the document's contents, for example, by displaying the title as a caption, or speaking it.

Titles may contain character entities (for accented characters, special characters, etc.), but may not contain other markup. Here is a sample document title:

```
<HTML>
<HEAD>
<TITLE>A study of population dynamics</TITLE>
... other head elements...
</HEAD>
<BODY>
... document body...
</BODY>
</HTML>
```

Related to the `TITLE` element is the `title` attribute.

Attribute definitions

`title` = *cdata*
: 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/neatstuff.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. Please consult this section 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.

Meta information

As this specification is being written, a number of approaches are being proposed for allowing authors to assign richer machine-readable information about documents and other network-accessible resources to an HTML document.

The current HTML specification allows authors to assign meta data to their documents as follows:

- Authors may cite an external profile where meta data properties are defined. For example, a profile might define properties that help search engines index documents, such as "author", "copyright", "keywords", etc. A profile is specified via the `profile` attribute of the `HEAD` element.
- Authors may set values for these properties. This may be done:
  1. From within a document, via the `META` element. Thus, the profile may define the name space of properties that can be set by the `META` element.
  2. From outside a document, by linking to meta data via the `LINK` element (see the section on link types). Thus, the profile may define the name space of relationship types that may be used by the `LINK` 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.

The META element

```
<!ELEMENT META - O EMPTY        -- Generic Metainformation -->
<!ATTLIST META
  %i18n;                          -- lang, dir, for use with content string --
  http-equiv  NAME       #IMPLIED  -- HTTP response header name  --
  name        NAME       #IMPLIED  -- metainformation name --
  content     CDATA      #REQUIRED -- associated information --
  scheme      CDATA      #IMPLIED  -- 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` = *cdata*
  This attribute specifies a property name.
`content` = *cdata*
  This attribute specifies a property's value.
`scheme` = *cdata*
  This attribute names a scheme to be used to interpret the property's value.
`http-equiv` = *cdata*
  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), `dir` (text direction)

The `META` element can be used to describe 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.

The `name` attribute specifies a property and the `content` attribute specifies the property's value. For example,

```
<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 synthesisers 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">
```

Here's another example: some user agents support the use of `META` to refresh the current page after a few seconds, perhaps replacing it with another page.

```
<META name="refresh" content="3,http://www.acme.com/intro.html">
```

The content is a number specifying the delay in seconds, followed by the URL to load when the time is up. This mechanism is generally used to show users a fleeting introductory page. However, since some user agents do not support this mechanism, authors should include content on the introductory page to allow users to navigate away from it (so they don't remain "stranded" on the introductory page).

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]-style header in the HTTP response. Please see the HTTP specification ([RFC2068]) 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.

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,

```
<META name="keywords" lang="en"
         content="vacation,Greece,sunshine">
<META name="keywords" lang="fr"
         content="vacances,Gr&egrave;ce,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.

META and PICS

The Platform for Internet Content Selection [PICS] 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))
 '>
<TITLE>..title goes here..</TITLE>
</HEAD>
<BODY>
...the body...
</BODY>
```

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.
- The default style sheet language.
- The document character encoding.

The following example specifies the character encoding 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 URL. User agents may use this URL 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 URL and, perform some activity based on the actual

definitions within the profile (e.g., validate 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="23 Jan 1997 16:05:31 GMT">
</HEAD>
```

As this specification is being written, it is common practice to use the date formats described in [RFC2068]. HTTP applications have historically allowed three different formats for the representation of date/time stamps:

```
Sun, 06 Nov 1994 08:49:37 GMT  ; RFC 822, updated by RFC 1123
Sunday, 06-Nov-94 08:49:37 GMT ; RFC 850, obsoleted by RFC 1036
Sun Nov  6 08:49:37 1994       ; ANSI C's asctime() format
```

According to [RFC2068], the first format is preferred. It represents a fixed-length subset of that defined by [RFC1123] (an update to [RFC822]). The second format is in common use, but is based on the obsolete [RFC850]date format and lacks a four-digit year. HTTP 1.1 clients must accept all of these forms, but only generate [RFC1123] format for use in HTTP headers. HTML user agents are expected to follow HTTP 1.1 in this regards, and in addition to provide support for the [ISO8601] date format, e.g. "1997-01-23T16:05:31+00:00". For more information, see the sections on the `INS` and `DEL` elements.

The `scheme` attribute is used to identify the expected format of the value of the `content` attribute, for cases when a property supports multiple formats. The values permitted for the `scheme` attribute depend on the property `name` and the `profile`.

The first `META` declaration in the following example refers to the Dewey Decimal System (dds) scheme. The second refers to the ISBN scheme.

```
<META scheme="dds" name="description"
         content="04.251 Supercomputers systems design">
<META scheme="ISBN"  name="identifier" content="0-8230-2355-9">
```

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

The **`BODY`** element

```
<!ENTITY % block "(%blocklevel | %inline)*">

<!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
  text  %Color   #IMPLIED
  link  %Color   #IMPLIED
  vlink %Color   #IMPLIED
  alink %Color   #IMPLIED
  ">

<!ELEMENT BODY O O  (%block) -(BODY) +(INS|DEL)>
<!ATTLIST BODY
  %attrs;                          -- %coreattrs, %i18n, %events --
  background  %URL       #IMPLIED  -- texture tile for document background --
  %bodycolors;                     -- bgcolor, text, link, vlink, alink --
  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` = *url*
: **Deprecated.**The value of this attribute is a URL that designates an image resource. The image generally tiles the background (for visual browsers).

`text` =*color*
: **Deprecated.**This attribute sets the foreground color for text (for visual browsers).

`link` = *color*
: **Deprecated.**This attribute sets the color of text marking unvisited hypertext links (for visual browsers).

`vlink` = *color*
: **Deprecated.**This attribute sets the color of text marking visited hypertext links (for visual browsers).

`alink` = *color*
: **Deprecated.**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)
- `lang` (language information), `dir` (text direction)
- `title` (element titles)
- `style` (inline style information)
- `bgcolor` (background color)
- `onload`, `onunload` (intrinsic events)
- `onclick`, `ondblclick`, `onmousedown`, `onmouseup`, `onmouseover`, `onmousemove`, `onmouseout`, `onkeypress`, `onkeydown`, `onkeyup` (intrinsic events)

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 are now the preferred way to specify a document's presentation, the presentational attributes of `BODY` have been deprecated. These attributes should only be used when you need to control the presentation for user agents that don't support style sheets.

The following line of HTML illustrates the use of the deprecated 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, fuschia when activated, and maroon once visited.

DEPRECATED EXAMPLE:

```
<HTML>
<HEAD>
 <TITLE>A study of population dynamics</TITLE>
</HEAD>
<BODY bgcolor="white" text="black"
  link="red" alink="fuschia" vlink="maroon">
  ... document body...
</BODY>
</HTML>
```

Using style sheets, the same effect could be accomplished as follows:

```
<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: fuschia }
 </STYLE>
</HEAD>
<BODY>
  ... document body...
</BODY>
</HTML>
```

Using linked style sheets gives you the flexibility to change the presentation without revising the document:

```
<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 for more information.*

Element identifiers: the id and class attributes

Attribute definitions

`id` = *name*
: This attribute assigns a document-wide name to a specific instance of an element. Values for `id` must be unique within a document. Furthermore, this attribute shares the same name space as the `name` attribute.

`class` = *cdata-list*
: This attribute assigns a class or set of classes to a specific instance of an element. Any number of elements may be assigned the same class name or names. They must be separated by white space characters.

The `id` and `class` attributes assign identifiers to an element instance.

An identifier specified by `id` must be unique within a document. A class name specified by `class` may be shared by several element instances. Class values should be chosen to distinguish the role of the element the class is associated with, e.g. note, example, warning.

These attributes can be used in the following ways:

- The `id` attribute may be used as a destination for hypertext links (see the section on anchors).
- Scripts can use the `id` attribute to reference a particular element.
- Style sheets can use the `id` attribute to apply a style to a particular element.
- The `id` attribute is used to identify `OBJECT` element declarations.
- Style sheets can use the `class` attribute to apply a style to a set of elements associated with this class, or to elements that occur as the children of such elements.
- Both `id` and `class` can be used for further processing purposes, e.g. for identifying fields when extracting data from HTML pages into a database, translating HTML documents into other formats, 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>
<STYLE
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.

Grouping elements: the `DIV` and `SPAN` elements

```
<!ELEMENT DIV - - %block>
<!ATTLIST DIV
  %attrs;                        -- %coreattrs, %i18n, %events --
  %align;                        -- align, text alignment --
  >
<!ELEMENT SPAN - - (%inline)*     -- generic language/style container -->
<!ATTLIST SPAN
  %attrs;                        -- %coreattrs, %i18n, %events --
  >
```

Start tag: ***required****, End tag:* ***required***

Attributes defined elsewhere

- `id`, `class` (document-wide identifiers)
- `lang` (language information), `dir` (text direction)
- `title` (element titles)
- `style` (inline style information)
- `align` (alignment)
- `onclick`, `ondblclick`, `onmousedown`, `onmouseup`, `onmouseover`, `onmousemove`, `onmouseout`, `onkeypress`, `onkeydown`, `onkeyup` (intrinsic events)

The `DIV` and `SPAN` elements, in conjunction with the `id` and `class` attributes, offer a generic mechanism for adding structure to documents. These are the only two HTML elements that do not add presentation to their enclosed content. Thus, by creating instances and classes of elements and applying style sheets to them, authors may specialize HTML according to their needs and tastes.

Suppose we wanted to generate a document from 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 tailor HTML to our own needs.

In this example, every client's last name belongs to the class "client-last-name", etc. We also assign a unique identifier to each client ("client-boyera", "client-lafon", etc.).

```
<DIV id="client-boyera" class="client">
<SPAN class="client-last-name">Last name:</SPAN> Boyera,
<SPAN class="client-first-name">First name:</SPAN> Stephane
<SPAN class="client-tel">Telephone:</SPAN> (212) 555-1212
<SPAN class="client-email">Email:</SPAN> sb@foo.org
</DIV>

<DIV id="client-lafon" class="client">
<SPAN class="client-last-name">Last name:</SPAN> Lafon,
<SPAN class="client-first-name">First name:</SPAN> Yves
<SPAN class="client-tel">Telephone:</SPAN> (617) 555-1212
<SPAN class="client-email">Email:</SPAN> yves@bar.com
</DIV>
```

Later, we may easily add style information to our document to fine tune the presentation of these database entries.

SPAN is an inline element and can be used within paragraphs, list items, etc. when you want assign class or language information to a group of words. SPAN cannot be used to group block-level elements. SPAN has no inherent effect on rendering until you apply a style, e.g., via a `style` attribute, or a linked style sheet.

DIV by contrast, is a block-level element. It can be used to group other block-level elements, but can't be used within paragraph elements. A DIV element following an unclosed P element will terminate that paragraph.

User agents generally place a line break before and after DIV elements, for instance:

```
<P>aaaaaaaaa<DIV>bbbbbbbbb</DIV><DIV>ccccc<P>ccccc</DIV>
```

This is typically rendered as:

```
aaaaaaaaa
bbbbbbbbb
ccccc

ccccc
```

Your user agent renders this as follows:

aaaaaaaaabbbbbbbbbccccc

ccccc

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;)*>
<!ATTLIST (%heading)
  %attrs;                          -- %coreattrs, %i18n, %events --
  %align;                          -- align, text alignment --
  >
```

*Start tag: **required**, End tag: **required***

Attributes defined elsewhere

- `id`, `class` (document-wide identifiers)
- `lang` (language information), `dir` (text direction)
- `title` (element titles)
- `style` (inline style information)
- `align` (alignment)
- `onclick`, `ondblclick`, `onmousedown`, `onmouseup`, `onmouseover`, `onmousemove`, `onmouseout`, `onkeypress`, `onkeydown`, `onkeyup` (intrinsic events)

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>
In this section, we discuss the lesser known forest elephants.
...this section continues...
<DIV class="subsection" id="forest-habitat" >
<H2>Habitat</H2>
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>
<STYLE>
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.

The `ADDRESS` element

```
<!ELEMENT ADDRESS - - ((%inline;) | P)*>
<!ATTLIST ADDRESS
  %attrs;                          -- %coreattrs, %i18n, %events --
  >
```

Start tag: ***required****, End tag:* ***required***

Attributes defined elsewhere

- `id`, `class` (document-wide identifiers)
- `lang` (language information), `dir` (text direction)
- `onclick`, `ondblclick`, `onmousedown`, `onmouseup`, `onmouseover`, `onmousemove`, `onmouseout`, `onkeypress`, `onkeydown`, `onkeyup` (intrinsic events)

For lack of a better place, we include the definition of the `ADDRESS` here. This element adds author and contact information to a document, e.g.,

```
<ADDRESS>
Newsletter editor<BR>
J. R. Brown<BR>
8723 Buena Vista, Smallville, CT 01234<BR>
Tel: +1 (123) 456 7890
</ADDRESS>
```

Language information and text direction

Contents

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.

Specifying the language of content: the `lang` attribute

Attribute definitions

`lang` = *language-code*
: Specifies the primary language of an element's text content. The value of this attribute is a language code as specified by [RFC1766]. Please consult this document for authoritative information on language codes. Whitespace is not allowed within the language-code. All language-codes are case-insensitive. The default language is "unknown".

Language information can be used to control rendering of a marked up document in a variety of ways. Some situations where this information helps include:

- Assisting search engines
- Speech synthesis
- Selecting glyph variants for high quality typography
- Choosing a set of quotation marks
- Resolving hyphenation ligatures, and spacing
- Spell checking and grammar checking

The `lang` attribute's value is a language code that identifies a natural language spoken, written, or otherwise conveyed by human beings for communication of information to other human beings. Computer languages are explicitly excluded from language codes.

[RFC1766] 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-cherokee": the Cherokee language spoken by some Native Americans.
- "x-pig-latin": the language "pig-latin" has not been registered with IANA, the organization that controls the namespace of language tags.

Two-letter primary codes are reserved for [ISO639] 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] country code.

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.
- An englobing element which has the `lang` attribute set (i.e., the `lang` attribute is inherited by nested elements).
- The HTTP "Content-Language" header, which may be set at the site level. For example:

  ```
  Content-Language: en-cockney
  ```

- Sources external to HTML (e.g., user agent default values, user preferences, HTTP headers).

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.

```
<HTML lang="fr">
<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>
```

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 "US".

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.

For artificial languages such as Elfish or Klingon, it would make sense to use the lang attribute to indicate the change from the language of the enclosing context. Until the successor to [RFC1766] defines a standard way to do this, one possibility is to use the x- prefix convention, e.g. x-elfish.

Specifying the direction of text: the `dir` attribute

Attribute definitions

`dir` = `LTR` | `RTL`
: Specifies the default direction for directionally weak or neutral text in the element's content (left-to-right or right-to-left) in this document. Possible values:
  - `LTR`: Left-to-right text.
  - `RTL`: Right-to-left text.

In addition to specifying the primary language of a document, authors may need to specify the default direction of pieces of text or the text in the entire document.

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

In the absence of local overrides, the default direction is inherited from enclosing elements.

Introduction to the bidirectional algorithm

The following example illustrates the expected behavior of the bidirectional algorithm.

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 "6".

Suppose the predominant language of the document containing this paragraph is English (left-to-right text). 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 (right-to-left direction), the correct presentation is:

```
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.

Inheritance of text direction information

The Unicode bidirectional algorithm requires an initial text direction. 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 primary text direction for an entire document, set the `dir` attribute on the `HTML` element.

For example:

```
<HTML dir="RTL">
...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.

Setting the direction of embedded text

The [UNICODE] bidirectional algorithm automatically reverses embedded character sequences according to their inherent directionality (as illustrated by the previous examples). However, 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. The above English sentence contains a Hebrew section extending from HEBREW2 through HEBREW4. 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 multiply 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 [ISO10646] 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 [ISO10646] 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.*

Overriding the bidirectional algorithm: the `BDO` element

```
<!ELEMENT BDO - - (%inline)*      -- I18N BiDi over-ride -->
<!ATTLIST BDO
  lang        NAME       #IMPLIED  -- [RFC1766] language value --
  dir         (ltr|rtl)  #REQUIRED -- directionality --
  >
```

Start tag: ***required****, End tag:* ***required***

Attributes defined elsewhere

- `lang` (language information), `dir` (text direction)

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 an English document containing the same text as before:

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

Suppose this sequence of characters is being read by a user agent from left-to-right (the byte stream begins with "e" and ends with "6"). The "e" in "english1" is to the left of "n", which is how authors tend to input English characters. However, the "H" in "HEBREW2" is to the left of "E", which may not be how authors of Hebrew create their documents. For example, the MIME standard ([RFC2045]) requires right-to-left character sequences in email to be ordered right-to-left in the byte stream. This conflicts with the [UNICODE] birectional algorithm, which expects Hebrew characters to be ordered left-to-right.

Thus, if "HEBREW4" in the above example were an excerpt from a Hebrew email message, it’s structure would actually be "4WERBEH". A user agent applying the bidirectional algorithm would thus display the characters in the wrong order.

The easiest solution in this case is to override the bidirectional algorithm by putting the Hebrew email excerpt in a `BDO` element, whose `dir` attribute is set to "LTR":

```
english1 HEBREW2 english3 <BDO dir="LTR">4WERBEH</BDO> english5 HEBREW6
```

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

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

The `BDO` 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 [ISO10646] formatting characters.

Bidirectionality and character encoding *According to [RFC1555] and [RFC1556], 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 full Unicode bidirectionality algorithm, conforming documents must be labeled as "iso-8859-8-e". Implicit bidirectionality is part of the full Unicode algorithm, so the values "iso-8859-8-i" may also be accepted, but should not be used.

*The value "iso-8859-8" defines 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] and [RFC1556], iso-8859-6 (Arabic) is **not** visual ordering.*

Support for character directionality and joining

Since ambiguities sometimes arise as to the directionality of certain characters (e.g., some situations in Arabic), the [UNICODE] specification includes characters to enable proper resolution. HTML 4.0 includes a set of named character entities that allows partial support of the Unicode bidirectional algorithm, plus some help with languages requiring contextual analysis for rendering.

The following DTD excerpt presents some of the directional entities:

```
<!ENTITY zwnj CDATA "&#8204;"--=zero width non-joiner-->
<!ENTITY zwj  CDATA "&#8205;"--=zero width joiner-->
<!ENTITY lrm  CDATA "&#8206;"--=left-to-right mark-->
<!ENTITY rlm  CDATA "&#8207;"--=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 disambiguate directionality of directionally neutral characters. For example, if a double quotation mark comes between an Arabic and a Latin 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] for more details.

Reversed character glyphs*: The bidirectional algorithm reverses the presentation of a well-defined set of characters such as parentheses (see [UNICODE], table 4-7). Except for these characters, bidirectionality processing leaves the shape of each glyph unaffected. Thus, if you wanted to display the word "MURDER" as it would be seen in a mirror (right-to-left character order and reversed glyphs), you could use a `BDO` element with the `dir` attribute to set the text direction to right-to-left order, e.g.,*

```
<BDO class="mirror" dir="rtl">MURDER</BDO>
```

and the class value "mirror" with a matching rule in the style sheet to select a special font that displays characters with the reversed glyphs.

The effect of style sheets on bidirectionality

In general, changing an element from being displayed in block from to inline or vice-versa due to a style sheet is straightforward. However, because the difference between block elements and inline elements is crucial for the bidirectional algorithm, special care must be taken.

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

When a block element that does not have a `dir` attribute is transformed to 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.

Undisplayable characters

User agents may not be able to render meaningfully all character values, for instance, because of the lack of an appropriate font, or because a character has a value which is inexpressible with the internal character encoding.

Because there are many different things that can be done in such a case, this document does not prescribe any specific behavior. Depending on the implementation, this may also be handled by the underlying display system and not the application itself. This specification recommends the following behavior for user agents:

1. Adopt a clearly visible, but unobtrusive mechanism to alert the user of missing resources.
2. If the user agent provides a numeric representation of missing characters, the hexadecimal (not decimal) form is preferable as this is the form used in character set standards (see [ERCS]).

Text

Contents

The following sections discuss issues surrounding the structuring of text. Elements that format text (alignment elements, font elements, style sheets, etc.) are discussed in later sections of the specification. Please consult the section on SGML for questions about character syntax.

White space

The SGML specification distinguishes between record start characters (line feeds) and record ends (carriage returns). On the Internet, some platforms use carriage return line feed pairs for line breaks, some use just line feeds, and others just carriage returns. As a result, HTML user agents should consider both isolated line feed and carriage return characters as line breaks, with carriage return line feed pairs treated as single line breaks.

A line break occurring immediately following a start tag should be discarded, as should a line break occurring immediately before an end tag. This applies to all HTML elements without exceptions. In addition, for all elements except PRE, a sequence of contiguous white space characters such as spaces, horizontal tabs, form feeds and line breaks, should be replaced by a single word space.

Since the notion of what word space is varies from script (written language) to script, user agents should collapse white space in script-sensitive ways. For example, in Latin scripts, a single word space is just a space (ASCII decimal 32), while in Thai it is a zero-width word separator. In Japanese and Chinese, a word space is ignored entirely.

These rules allow authors to use white space to lay out their markup as desired, clarifying the source HTML with white space that will not be rendered by a user agent.

For instance, the following source HTML:

```
<P>
  This example shows a paragraph and a list.
</P>

<UL>
  <LI>
    This is the <EM>first</EM> item
  </LI>

  <LI>
    This is the <EM>second</EM> item
  </LI>
</UL>
```

may be rewritten (by omitting end tags) and laid out differently (by using less white space):

```
<P>This example shows a paragraph and a list.

<UL>
  <LI>This is the <EM>first</EM> item
  <LI>This is the <EM>second</EM> item
</UL>
```

but should be rendered identically by a user agent.

The `PRE` element is used for preformatted text, where white space is significant. The `PRE` element is described below. *Word space processing can and should be done even in the absence of language information specified by the* `lang` *attribute. This is a script issue, not a language issue.*

Structured text

Phrasal elements: `EM`, `STRONG`, `DFN`, `CODE`, `SAMP`, `KBD`, `VAR`, `CITE`, and `ACRONYM`

```
<!ENTITY % phrase "EM | STRONG | DFN | CODE |
                   SAMP | KBD | VAR | CITE | ACRONYM">
<!ELEMENT (%font|%phrase) - - (%inline)*>
<!ATTLIST (%font|%phrase)
  %attrs;                          -- %coreattrs, %i18n, %events --
  >
```

Start tag: ***required****, End tag:* ***required***

Attributes defined elsewhere

- `id`, `class` (document-wide identifiers)
- `lang` (language information), `dir` (text direction)
- `title` (element titles)
- `style` (inline style information)

- `onclick`, `ondblclick`, `onmousedown`, `onmouseup`, `onmouseover`, `onmousemove`, `onmouseout`, `onkeypress`, `onkeydown`, `onkeyup` (intrinsic events)

Phrasal elements add structural information to text fragments. The usual meanings of phrasal elements are following:

EM:
: Indicates emphasis.

STRONG:
: Indicates stronger emphasis.

CITE:
: Cites a reference or other source.

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.

ACRONYM:
: Indicates an acronym (e.g., WWW, HTTP, URL, etc.).

`EM` and `STRONG` are useful in general to indicate emphasis. The other phrasal elements have particular significance in technical documents. These examples illustrate the rendering of some of the textual markup elements:

```
"More information can be found in <CITE>[ISO-0000]</CITE>."

"Please refer to the following reference number in future
correspondence: <STRONG>1-234-55</STRONG>"
```

The `ACRONYM` element allows authors to clearly indicate a sequence of characters that compose an acronym (e.g., "WWW", "FNAC", "IRS", etc.). The ability to identify acronyms is useful to spell checkers, speech synthesizers, and other user agents and tools.

The content of the `ACRONYM` element specifies the acronym itself. The `title` attribute may be used to provide the text to which the acronym expands. Here are some sample acronym definitions:

```
<ACRONYM title="World Wide Web">WWW</ACRONYM>
<ACRONYM
   lang="fr"
   title="Soci&eacute;t&eacute; Nationale de Chemins de Fer">
   SNCF
</ACRONYM>
```

The presentation of phrasal elements depends on the user agent. Generally, visual user agents present `EM` text in italics and `STRONG` text in bold font. Speech synthesiser agents might change the synthesis parameters, such as volume, pitch and rate accordingly. Acronyms are generally spoken by pronouncing the individual letters separately.

Note: This version of HTML doesn't include special markup for abbreviations. We recommend that speech synthesizers use client-side dictionaries to expand any abbreviations found in the document. For specialized vocabularies, `LINK` *elements in the document head can be used to reference suitable dictionaries.*

Quotations: The `BLOCKQUOTE` and `Q` elements

```
<!ELEMENT BLOCKQUOTE - - %block>
<!ATTLIST BLOCKQUOTE
  %attrs;                         -- %coreattrs, %i18n, %events --
  cite        %URL       #IMPLIED  -- URL for source document or msg --
  >
<!ELEMENT Q - - (%inline)*>
<!ATTLIST Q
  %attrs;                         -- %coreattrs, %i18n, %events --
  cite        %URL       #IMPLIED  -- URL for source document or msg --
  >
```

Start tag: ***required****, End tag:* ***required***

Attribute definitions

`cite` = *url*
: The value of this attribute is a URL that designates a source document or message. This attribute is intended to give information about the source from which the quotation was borrowed.

Attributes defined elsewhere

- `id`, `class` (document-wide identifiers)
- `lang` (language information), `dir` (text direction)
- `title` (element titles)
- `style` (inline style information)
- `onclick`, `ondblclick`, `onmousedown`, `onmouseup`, `onmouseover`, `onmousemove`, `onmouseout`, `onkeypress`, `onkeydown`, `onkeyup` (intrinsic events)

These two elements designate quoted text. `BLOCKQUOTE` is for long quotations and `Q` is intended for short quotations that don't require paragraph breaks.

This example formats an excerpt from "The Two Towers", by J.R.R. Tolkien, as a blockquote.

```
<BLOCKQUOTE cite="http://www.mycom.com/tolkien/twotowers.html">
They went in single file, running like hounds on a strong scent,
and an eager light was in their eyes. Nearly due west the broad
swath of the marching Orcs tramped its ugly slot; the sweet grass
of Rohan had been bruised and blackened as they passed.
</BLOCKQUOTE>
```

Visual user agents generally render BLOCKQUOTE as an indented block.

Quotation marks*It is recommended that style sheets provide a way to insert quotation marks before and after a quotation delimited by* Q *or* BLOCKQUOTE *in a manner appropriate to the current language context (see the* lang *attribute) and the degree of nesting of quotations.*

However, as some authors have used BLOCKQUOTE *merely as a mechanism to indent text, in order to preserve the intention of the authors, it is recommended that user agents* **not** *insert quotation marks in the default style.*

Furthermore, if authors include quotation marks in a Q *or* BLOCKQUOTE *element, user agents should not insert additional quotation marks.*

The usage of BLOCKQUOTE *to indent text is deprecated in favor of style sheets.*

Subscripts and superscripts: the SUB and SUP elements

```
<!-- subscripts and superscripts -->
<!ELEMENT (SUB|SUP) - - (%inline)*>
<!ATTLIST (SUB|SUP)
  %attrs;                        -- %coreattrs, %i18n, %events --
  >
```

Start tag: ***required****, End tag:* ***required***

Attributes defined elsewhere

- id, class (document-wide identifiers)
- lang (language information), dir (text direction)
- title (element titles)
- style (inline style information)
- onclick, ondblclick, onmousedown, onmouseup, onmouseover, onmousemove, onmouseout, onkeypress, onkeydown, onkeyup (intrinsic events)

Many scripts (e.g., French) require superscripts or subscripts for proper rendering. The SUB and SUP elements should be used to markup text in these cases.

Here, we use SUP to raise the "lle" in the French "M$^{\text{lle}}$ Dupont":

```
M<sup>lle</sup> Dupont
```

Lines and Paragraphs

Authors traditionally divide their thoughts and arguments into sequences of paragraphs. The organization of information into paragraphs is not affected by how the paragraphs are presented: paragraphs that are double-justified contain the same thoughts as those that are left-justified.

The HTML markup for *defining* a paragraph is straightforward: the P element defines a paragraph.

The visual presentation of paragraphs is not so simple. A number of issues, both stylistic and technical, must be addressed:

- Treatment of white space
- Line breaking and word wrapping
- Justification
- Hyphenation
- Written language conventions and text directionality
- Formatting of paragraphs with respect to surrounding content

We address these questions below. Paragraph alignment and floating objects are discussed later in this document.

Paragraphs: the P element

```
<!ELEMENT P - O (%inline)*>
<!ATTLIST P
  %attrs;                          -- %coreattrs, %i18n, %events --
  %align;                          -- align, text alignment --
  >
```

Start tag: ***required****, End tag:* ***optional***

Attributes defined elsewhere

- id, class (document-wide identifiers)
- lang (language information), dir (text direction)
- title (element titles)
- style (inline style information)
- align (alignment)
- onclick, ondblclick, onmousedown, onmouseup, onmouseover, onmousemove, onmouseout, onkeypress, onkeydown, onkeyup (intrinsic events)

The P element represents a paragraph. It cannot contain block-level elements (including P itself). You can omit the end tag, which is then implied by the next block-level start tag. It is also implied by the end tag of the element that encloses the P element. For example, the following two paragraphs:

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

may be rewritten without their end tags:

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

since both are implicitly ended by the block elements that follow them. 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 `P` element.

Empty `P` elements are bad form and should be ignored by the renderer.

Visual rendering of paragraphs

How paragraphs are rendered visually depends on the user agent. Paragraphs are usually rendered flush left with a ragged right margin. Other defaults are appropriate for right-to-left scripts.

HTML user agents have traditionally rendered paragraphs with white space before and after, e.g.,

```
  At the same time, there began to take form a system of numbering,
  the calendar, hieroglyphic writing, and a technically advanced
  art, all of which later influenced other peoples.

  Within the framework of this gradual evolution or cultural
  progress the Preclassic horizon has been divided into Lower,
  Middle and Upper periods, to which can be added a transitional
  or Protoclassic period with several features that would later
  distinguish the emerging civilizations of Mesoamerica.
```

This contrasts with the style used in novels which indents the first line of the paragraph and uses the regular line spacing between the line of the last paragraph and the first line of the next, e.g.,

```
     At the same time, there began to take form a system of
  numbering, the calendar, hieroglyphic writing, and a technically
  advanced art, all of which later influenced other peoples.
     Within the framework of this gradual evolution or cultural
  progress the Preclassic horizon has been divided into Lower,
  Middle and Upper periods, to which can be added a transitional
  or Protoclassic period with several features that would later
  distinguish the emerging civilizations of Mesoamerica.
```

Following the precedent set by the NCSA Mosaic browser in 1993, user agents generally don't justify both margins, in part because it's hard to do this effectively without sophisticated hyphenation routines. The advent of style sheets, and antialiased fonts with subpixel positioning promises to offer richer choices to HTML authors than previously possible.

Style sheets provide rich control over the size and style of a font, the margins, space before and after a paragraph, the first line indent, justification and many other details. The user agent's default style sheet renders `P` elements in a familar form, as illustrated above. You could in principle override this to render paragraphs without the breaks that conventionally distinguish successive paragraphs, but this would be confusing to readers and as a rule bad practice.

By convention, visual HTML user agents wrap text lines to fit within the available margins. Wrapping algorithms depend on the script being formatted.

In Western scripts, for example, text should only be wrapped at white space. Early user agents incorrectly wrapped lines at the beginning (or end) of elements, which resulted in dangling punctuation. For example,

```
A statue of the <a href="cih78">Cihuateteus</a>, who are patron ...
```

Wrapping the line at the end of the anchor tag causes the comma to be stranded at the beginning of the next line:

```
A statue of the Cihuateteus
, who are patron goddesses ...
```

This is an error, since there was no white space at that point in the markup.

Controlling line breaks

It is possible to force or forbid a line break in HTML.

Forcing a line break: the `BR` element

```
<!ELEMENT BR - O EMPTY          -- forced line break -->
<!ATTLIST BR
  %coreattrs;                     -- id, class, style, title --
  clear (left|all|right|none) none -- control of text flow --
  >
```

Start tag: ***required****, End tag:* ***forbidden***

Attributes defined elsewhere

- `id`, `class` (document-wide identifiers)
- `title` (element titles)
- `style` (inline style information)
- `clear` (alignment and floating objects)

The `BR` element forcibly breaks (ends) the current line of text.

For visual user agents, the `clear` attribute can be used to determine whether markup following the `BR` element flows around images and other objects floated to the left or right margin, or whether it starts after the bottom of such objects. Further details are given in the section on alignment and floating objects. Authors are advised to use style sheets to control text flow around floating images and other objects. The `clear` attribute, along with other HTML presentation attributes and tags, is only appropriate when you need to consider user agents that don’t support style sheets.

With respect to bidirectional formatting, the `BR` element should be treated in the same way as a Unicode LINE SEPARATOR character.

Prohibiting a line break

Sometimes you will want to prevent a line break from occurring between two words. The   entity (,) acts as a space where user agents should not cause a line break.

Hyphenation

In HTML, there are two types of hyphens: the plain hyphen and the soft hyphen. The plain hyphen should be interpreted by a user agent as just another character. The soft hyphen tells the user agent where a line break can occur.

Those browsers that interpret soft hyphens must observe the following semantics: If a line is broken at a soft hyphen, a hyphen character must be displayed at the end of the first line. If a line is not broken at a hyphen, the user agent must not display a hyphen character. For operations such as searching and sorting, the soft hyphen should always be ignored.

In HTML, the plain hyphen is represented by the "-" character (-, -). The soft hyphen is represented by the named character entity (,)

Preformatted text: The `PRE` element

```
<!ENTITY % pre.exclusion "IMG|BIG|SMALL|SUB|SUP|FONT">

<!ELEMENT PRE - - (%inline)* -(%pre.exclusion)>
<!ATTLIST PRE
  %attrs;                          -- %coreattrs, %i18n, %events --
  width       NUMBER     #IMPLIED
  >
```

*Start tag: **required**, End tag: **required***

Attribute definitions

`width` = *integer*
: This attribute provides a hint to visual user agents about the desired width of the formatted block. The user agent can use this information to select an appropriate font size or to indent the content appropriately. The desired width is expressed in number of characters. This attribute is not widely supported currently.

Attributes defined elsewhere

- `id`, `class` (document-wide identifiers)
- `lang` (language information), `dir` (text direction)
- `title` (element titles)
- `style` (inline style information)
- `onclick`, `ondblclick`, `onmousedown`, `onmouseup`, `onmouseover`, `onmousemove`, `onmouseout`, `onkeypress`, `onkeydown`, `onkeyup` (intrinsic events)

The `PRE` element tells visual user agents that the enclosed text is "preformatted". Visual user agents must treat preformatted text as follows:

- They may leave white space intact.
- They may render text with a fixed-pitch font.
- They may disable automatic word wrap.
- They must not disable bidirectional processing.

Note that the SGML standard requires that the parser remove a newline immediately following the start tag or immediately preceding the end tag.

The DTD fragment above indicates which elements may not appear within a `PRE` declaration. This is the same as in HTML 3.2, and is intended to preserve constant line spacing and column alignment for text rendered in a fixed pitch font. Authors are discouraged from altering this behavior through style sheets.

The following example shows a preformatted verse from Shelly's poem *To a Skylark*:

```
<PRE>
       Higher still and higher
         From the earth thou springest
       Like a cloud of fire;
         The blue deep thou wingest,
And singing still dost soar, and soaring ever singest.
</PRE>
```

Here is the same verse as rendered by your user agent:

```
       Higher still and higher
         From the earth thou springest
       Like a cloud of fire;
         The blue deep thou wingest,
And singing still dost soar, and soaring ever singest.
```

The horizontal tab character
The horizontal tab character (encoded in [UNICODE], US ASCII, and [ISO88591] as decimal 9) is usually interpreted by visual user agents as the smallest non-zero number of spaces necessary to line characters up along tab stops that are every 8 characters. We strongly discourage using horizontal tabs in preformatted text since it is common practice, when editing, to set the tab-spacing to other values, leading to misaligned documents.

Marking document changes: The INS and DEL elements

```
<!-- INS/DEL are handled by inclusion on BODY -->
<!ELEMENT (INS|DEL) - - (%inline)* -- inserted/deleted text -->
<!ATTLIST (INS|DEL)
  %attrs                          -- %coreattrs, %i18n, %events --
  cite        %URL      #IMPLIED   -- info on reason for change --
  datetime    CDATA     #IMPLIED   -- when changed: ISO date format --
  >
```

*Start tag: **required**, End tag: **required***

Attribute definitions

`cite` = *url*
: The value of this attribute is a URL that designates a source document or message. This attribute is intended to point to information explaining why a document was changed.

`datetime` = *cdata*
: The value of this attribute specifies the date and time when the change was made. This value must have a format as specified in [ISO8601] and limited by the profile defined in the section below on dates and times.

Attributes defined elsewhere

- `id`, `class` (document-wide identifiers)
- `lang` (language information), `dir` (text direction)
- `title` (element titles)
- `style` (inline style information)
- `onclick`, `ondblclick`, `onmousedown`, `onmouseup`, `onmouseover`, `onmousemove`, `onmouseout`, `onkeypress`, `onkeydown`, `onkeyup` (intrinsic events)

`INS` and `DEL` are used to markup sections of the document that have been inserted or deleted since a previous version of a document (e.g., in draft legislation where law makers need to view the changes).

These two elements are unsual for HTML in that they are neither block-level nor inline elements. They may contain one or more words within a paragraph or englobe one or more block-level elements such as paragraphs, lists and tables.

User agents may render inserted and deleted text in ways that make the change obvious. For instance, inserted text may appear in a special font, deleted text may not be shown at all or be shown as struck-through or with special markings, etc.

User agents that do not recognize the `DEL` element must render that element’s content nonetheless.

Date and time format

[ISO8601] allows many options and variations in the representation of dates and times. This specification defines a specific format which is one of those allowed by [ISO8601].

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).

`+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, to indicate the beginning of the time element, as specified in [ISO8601]

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).

Both of the following examples correspond to November 5, 1994, 8:15:30 am, US Eastern Standard Time.

```
1994-11-05T13:15:30Z
1994-11-05T08:15:30-05:00
```

Used with `INS`, this gives:

```
<INS datetime="1994-11-05T08:15:30-05:00">
I added this on November 5th
</INS>
```

Lists

Contents

Unordered (UL) and ordered (OL) lists

```
<!ENTITY % ULStyle "disc|square|circle">

<!ELEMENT UL - -  (LI)+>
<!ATTLIST UL                    -- unordered lists --
  %attrs;                          -- %coreattrs, %i18n, %events --
  type        (%ULStyle) #IMPLIED  -- bullet style --
  compact     (compact)  #IMPLIED  -- reduced interitem spacing --
  >
<!ENTITY % OLStyle "CDATA"      -- constrained to: [1|a|A|i|I] -->

<!ELEMENT OL - -  (LI)+>
<!ATTLIST OL -- ordered lists --
  %attrs;                          -- %coreattrs, %i18n, %events --
  type        %OLStyle   #IMPLIED  -- numbering style --
  compact     (compact)  #IMPLIED  -- reduced interitem spacing --
  start       NUMBER     #IMPLIED  -- starting sequence number --
  >
```

*Start tag: **required**, End tag: **required***

```
<!-- The type attribute can be used to change the bullet style
     in unordered lists and the numbering style in ordered lists -->

<!ENTITY % LIStyle "CDATA" -- constrained to: "(%ULStyle|%OLStyle)" -->

<!ELEMENT LI - O %block -- list item -->
<!ATTLIST LI
  %attrs;                          -- %coreattrs, %i18n, %events --
  type        %LIStyle   #IMPLIED  -- list item style --
  value       NUMBER     #IMPLIED  -- reset sequence number --
  >
```

*Start tag: **required**, End tag: **optional***

Attribute definitions

type = *style-information*
: This attribute sets the style of a list item. Currently available values are intended for visual user agents. Possible values are described below.

`start` = *integer*
: For `OL` only. This attribute specifies the starting number of the first item in an ordered list. The default starting number is one.

`value` = *integer*
: For `LI` only. This attribute sets the current number of a list element in an ordered list to a new integer value.

`compact`
: **Deprecated.** When set, this boolean attribute gives a hint to visual user agents to render the list in a more compact way.

Attributes defined elsewhere

- `id`, `class` (document-wide identifiers)
- `lang` (language information), `dir` (text direction)
- `title` (element titles)
- `style` (inline style information)
- `onclick`, `ondblclick`, `onmousedown`, `onmouseup`, `onmouseover`, `onmousemove`, `onmouseout`, `onkeypress`, `onkeydown`, `onkeyup` (intrinsic events)

Ordered and unordered lists are identical except that visual user agents number ordered list items. User agents may present those numbers in a variety of ways. Unordered list items are not numbered.

Both types of lists are made up of sequences of list items defined by the `LI` element (whose end tag is generally omitted).

This example illustrates the basic structure of a list.

```
<UL>
   <LI> ... first list item...
   <LI> ... second list item...
   ...
</UL>
```

Lists may also be nested:

```
<UL>
     <LI> ... Level one, number one...
     <OL>
        <LI> ... Level two, number one...
        <LI> ... Level two, number two...
        <OL start="10">
           <LI> ... Level three, number one...
        </OL>
        <LI> ... Level two, number three...
     </OL>
     <LI> ... Level one, number two...
</UL>
```

Details about number order. *In ordered lists, it is not possible to continue list numbering automatically from a previous list or to hide numbering of some list items. However, you can reset the number of a list item by setting its* `value` *attribute. Numbering continues from the new value for subsequent list items. For example:*

```
<ol>
<li value="30"> makes this list item number 30.
<li value="40"> makes this list item number 40.
<li> makes this list item number 41.
</ol>
```

Lists formatted by visual user agents

The following description refers to the behavior of some current visual user agents when formatting lists. Style sheets allow better control of list formatting (e.g., for numbering, language-dependent conventions, etc.).

Visual browsers usually present nested lists indented with respect to the current level of indentation.

For both `OL` and `UL`, the `type` attribute specifies rendering options for visual user agents.

For the `UL` element, possible values for the `type` attribute are `disc`, `square`, and `circle`. The default value depends on the level of nesting of the current list.

How each value is presented depends on the user agent. User agents should attempt to present a "disc" as a small filled-in circle, a "circle" as a small circle outline, and a "square" as a small square outline.

Your user agent displays them as follows (the bullet glyph in the line may or may not vary):

- is produced by the value "disc"
- is produced by the value "square"
- is produced by the value "circle"

For the `OL` element, possible values for the `type` attribute are summarized in the table below:

Type	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, ...

Definition lists: the `DL`, `DT`, and `DD` elements

```
<!-- definition lists - DT for term, DD for its definition -->

<!ELEMENT DL - -  (DT|DD)+>
<!ATTLIST DL
  %attrs;                          -- %coreattrs, %i18n, %events --
  compact     (compact)  #IMPLIED  -- reduced interitem spacing --
  >
```

*Start tag: **required**, End tag: **required***

```
<!ELEMENT DT - O  (%inline)*>
<!ELEMENT DD - O  %block>
<!ATTLIST (DT|DD)
  %attrs                          -- %coreattrs, %i18n, %events --
  >
```

*Start tag: **required**, End tag: **optional***

Attributes defined elsewhere

- `id`, `class` (document-wide identifiers)
- `lang` (language information), `dir` (text direction)
- `title` (element titles)
- `style` (inline style information)
- `onclick`, `ondblclick`, `onmousedown`, `onmouseup`, `onmouseover`, `onmousemove`, `onmouseout`, `onkeypress`, `onkeydown`, `onkeyup` (intrinsic events)

Definition lists vary only slightly from other types of lists in that list items consist of two parts: an initial label and a description. The label part is initiated by the `DT` element and may only contain marked up text. The description begins with the `DD` element and may contain block-level content.

Here is a sample definition list.

```
<DL>
    <DT> <em>Daniel</em>
    <DD> Born in France, Daniel's favorite food is foie gras.
         <P> In this paragraph, we'll discuss Daniel's
         harem: Pascale, Audrey, Laurie, and Alice.
    <DT> <em>Tim</em>
    <DD> Born in New York, Tim's favorite food is ice cream.
  </DL>
```

The rendering of a definition list depends on the user agent. Your user agent renders this example as follows:

Daniel
: Born in France, Daniel's favorite food is foie gras.

In this paragraph, we'll discuss Daniel's harem: Pascale, Audrey, Laurie, and Alice.

Tim

Born in New York, Tim's favorite food is ice cream.

The `DIR` and `MENU` elements

DIR and MENU are deprecated

```
<!ELEMENT (DIR|MENU) - -  (LI)+ -(%blocklevel)>
<!ATTLIST DIR
  %attrs;                          -- %coreattrs, %i18n, %events --
  compact     (compact)  #IMPLIED
  >
<!ATTLIST MENU
  %attrs;                          -- %coreattrs, %i18n, %events --
  compact     (compact)  #IMPLIED
  >
```

*Start tag: **required**, End tag: **required***

Attributes defined elsewhere

- `id`, `class` (document-wide identifiers)
- `lang` (language information), `dir` (text direction)
- `title` (element titles)
- `style` (inline style information)
- `onclick`, `ondblclick`, `onmousedown`, `onmouseup`, `onmouseover`, `onmousemove`, `onmouseout`, `onkeypress`, `onkeydown`, `onkeyup` (intrinsic events)

The `DIR` element was designed to be used for creating multicolumn directory lists. The `MENU` element was designed to be used for single column menu lists. Both elements have the same structure as `UL`, just different rendering. In practice, a user agent will render a `DIR` or `MENU` list exactly as a `UL` list.

We strongly recommend using `UL` instead of these elements.

Tables

Contents

The HTML table model allows users to organize data in complex tabular structures. Tables can include lists, paragraphs, forms, figures, preformatted text, and other tables.

In this table model, rows and columns may be grouped together. This grouping conveys structural information about the table and may be rendered by user agents in ways to emphasize this structure.

Row groups are particularly useful in large tables. Intelligent visual user agents may allow scrolling of a table body while preserving the head and foot information on the screen. Similarly, when long tables are printed, the head and foot information may be repeated on each page that contains table data.

Note: This specification includes more detailed information about tables in sections on table design rationale and implementation issues.

Table structure

An HTML table has the following structure:

- An optional caption.
- One or more groups of rows. Each row group consists of an optional head section, an optional foot section, and a series of rows.
- One or more groups of columns.
- Each row consists of one or more cells.
- Each cell may contain either header information (meant to describe the nature of data in the column or row) or data. A cell may span more than one row or column.

The **`TABLE`** element

```
<!ELEMENT TABLE - - (CAPTION?, (COL*|COLGROUP*), THEAD?, TFOOT?, TBODY+)>
<!ATTLIST TABLE                        -- table element --
  %attrs;                              -- %coreattrs, %i18n, %events --
  align       %TAlign;    #IMPLIED  -- table position relative to window --
  bgcolor     %Color      #IMPLIED  -- background color for cells --
  width       CDATA       #IMPLIED  -- table width relative to window --
  cols        NUMBER      #IMPLIED  -- used for immediate display mode --
  border      CDATA       #IMPLIED  -- controls frame width around table --
  frame       %TFrame;    #IMPLIED  -- which parts of table frame to include --
  rules       %TRules;    #IMPLIED  -- rulings between rows and cols --
  cellspacing CDATA       #IMPLIED  -- spacing between cells --
  cellpadding CDATA       #IMPLIED  -- spacing within cells --
  >
```

Start tag: ***required****, End tag:* ***required***

Attribute definitions

`align` = `left`|`center`|`right`
: This attribute specifies the position of the table with respect to the document. Possible values:
  - `left`: The table is to the left of the document.
  - `center`: The table is to the center of the document.
  - `right`: The table is to the right of the document.

`width` = *length*
: This attribute specifies the desired width of entire table for visual user agents. In the absence of this attribute, table width is determined by the user agent.

`cols` = *integer*
: This attribute specifies the number of columns for the table. When specified, this attribute helps visual user agents render the table as soon as it receives incoming data, rather than having to wait for the entire table to determine the number of columns for certain.

Attributes defined elsewhere

- `id`, `class` (document-wide identifiers)
- `lang` (language information), `dir` (text direction)
- `title` (element titles)
- `style` (inline style information)
- `onclick`, `ondblclick`, `onmousedown`, `onmouseup`, `onmouseover`, `onmousemove`, `onmouseout`, `onkeypress`, `onkeydown`, `onkeyup` (intrinsic events)
- `bgcolor` (background color)
- `frame`, `rules`, `border` (borders and rules)
- `cellspacing`, `cellpadding` (cell margins)

The `TABLE` element contains all other elements that specify caption, rows, content, and formatting.

Calculating the number of rows and columns in a table

The number of rows in a table is equal to the number of TR elements it contains. User agents should ignore rows that are implied by cells spanning rows beyond this number.

There are several ways to determine the number of columns:

- Scan each row in turn to compute the minimum number of columns needed (taking column spans into account). If the column count for the table exceeds the number of cells in a given row (including spanned rows), the end of that row is padded with empty cells. The "end" of a row depends on the directionality of the table.
- Count the number of columns as specified by COL and COLGROUP elements which can only occur at the start of the table (after the optional CAPTION.
- Use the cols attribute on the TABLE element. This is the weakest method as you don’t get any additonal information on column widths. This may not matter though if you use style sheets to specify widths.

User agents can assume that the table in this example has three columns.

```
<TABLE cols="3">
...the rest of the table...
</TABLE>
```

If the number of columns in a table is not specified by the cols attribute, a visual user agent may wait for the entire table to arrive before beginning rendering. In general waiting until the end of the table allows the number of columns and their widths to be determined without the need for a redisplay. Setting the cols attribute acts as a hint to visual user agents to display tables as each row is received. Authors are recommended to use the COL and COLGROUP elements to specify column properties rather than using the cols attribute.

Table directionality

The directionality of a table is specified by the dir attribute for the TABLE element. For a left-to-right table (the default), column one is on the left side of the table and row one is at the top. For a right-to-left table, column one is one the right side and row one is at the top.

Similarly, for left-to-right tables (the default), extra row cells are added to the right of the table, and for right-to-left tables, extra cells are added to the left side.

When set for the TABLE element, the dir attribute also affects the direction of text within table cells (since the dir attribute is inherited by block-level elements).

To specify a right-to-left table, set the dir attribute as follows:

```
<TABLE dir="RTL">
...the rest of the table...
</TABLE>
```

The direction of text in individual cells can be changed by setting the `dir` attribute in element that defines the cell. Please consult the section on bidirectional text for more information on text direction issues.

Table Captions: The `CAPTION` element

```
<!ELEMENT CAPTION - - (%inline;)+>
<!ENTITY % CAlign "(top|bottom|left|right)">

<!ATTLIST CAPTION              -- table caption --
  %attrs;                        -- %coreattrs, %i18n, %events --
  align       %CAlign;   #IMPLIED  -- relative to table --
  >
```

Start tag: ***required****, End tag:* ***required***

Attribute definitions

`align` = `top|bottom|left|right`
This attribute specifies the position of the caption with respect to the table. Possible values:
- `top`: The caption is above the table. This is the default value.
- `bottom`: The caption is below the table.
- `left`: The caption is to the left of the table.
- `right`: The caption is to the right of the table.

Attributes defined elsewhere

- `id`, `class` (document-wide identifiers)
- `lang` (language information), `dir` (text direction)
- `title` (element titles)
- `style` (inline style information)
- `onclick`, `ondblclick`, `onmousedown`, `onmouseup`, `onmouseover`, `onmousemove`, `onmouseout`, `onkeypress`, `onkeydown`, `onkeyup` (intrinsic events)

When present, the `CAPTION` element's text should describe the nature of the table. The `CAPTION` element must come immediately after the `TABLE` start tag.

For instance,

```
<TABLE cols="3">
<CAPTION>Cups of coffee consumed by each senator</CAPTION>
...the rest of the table...
</TABLE>
```

Groups of rows: the `THEAD`, `TFOOT`, and `TBODY` elements

```
<!ELEMENT THEAD - O (TR+)>
<!ELEMENT TFOOT - O (TR+)>
```

*Start tag: **required**, End tag: **optional***

```
<!ELEMENT TBODY O O (TR+)>
```

*Start tag: **optional**, End tag: **optional***

```
<!ATTLIST (THEAD|TBODY|TFOOT)   -- table section --
  %attrs;                          -- %coreattrs, %i18n, %events --
  %cellhalign;                     -- horizontal alignment in cells --
  %cellvalign;                     -- vertical alignment in cells --
  >
```

Attributes defined elsewhere

- `id`, `class` (document-wide identifiers)
- `lang` (language information), `dir` (text direction)
- `title` (element titles)
- `style` (inline style information)
- `onclick`, `ondblclick`, `onmousedown`, `onmouseup`, `onmouseover`, `onmousemove`, `onmouseout`, `onkeypress`, `onkeydown`, `onkeyup` (intrinsic events)
- `align`, `char`, `charoff`, `valign` (cell alignment)

A table must contain at least one row group. Each row group is divided into three sections: head, body, and foot. The head and foot sections are optional. The `THEAD` element defines the head, the `TFOOT` element defines the foot, and the `TBODY` element defines the body.

When present, each `THEAD`, `TFOOT`, and `TBODY` instance must contain one or more rows (see `TR`).

This example illustrates the order and structure of table heads, feet, and bodies.

```
<TABLE>
<THEAD>
     <TR> ...header information...
</THEAD>
<TFOOT>
     <TR> ...footer information...
</TFOOT>
<TBODY>
     <TR> ...first row of block one data...
     <TR> ...second row of block one data...
</TBODY>
<TBODY>
     <TR> ...first row of block two data...
     <TR> ...second row of block two data...
     <TR> ...third row of block two data...
</TBODY>
</TABLE>
```

`TFOOT` must appear before `TBODY` within a `TABLE` definition so that user agents can render the foot before receiving all of the (potentially numerous) rows of data.

Optional row group tags

- When a table contains only one body and no head or foot sections, the TBODY start and end tags may be omitted.
- When a table block contains a head, the start tag of the THEAD element is required. The end tag may be omitted when a TFOOT or TBODY start tag follows.
- When a table block contains a foot, the start tag of the TFOOT element is required. The end tag may be omitted when a THEAD or TBODY start tag follows.

Conforming user agent parsers must obey these rules for reasons of backward compatibility.

The table of the previous example could be shortened by removing certain end tags.

```
<TABLE>
<THEAD>
     <TR> ...header information...
<TFOOT>
     <TR> ...footer information...
<TBODY>
     <TR> ...first row of block one data...
     <TR> ...second row of block one data...
<TBODY>
     <TR> ...first row of block two data...
     <TR> ...second row of block two data...
     <TR> ...third row of block two data...
</TABLE>
```

Groups of columns: the COLGROUP and COL elements

The COLGROUP element

```
<!ELEMENT COLGROUP - O (col*)>
<!ATTLIST COLGROUP
  %attrs;                          -- %coreattrs, %i18n, %events --
  span        NUMBER     1         -- default number of columns in group --
  width       CDATA      #IMPLIED  -- default width for enclosed COLs --
  %cellhalign;                     -- horizontal alignment in cells --
  %cellvalign;                     -- vertical alignment in cells --
  >
```

*Start tag: **required**, End tag: **optional***

Attribute definitions

span = *integer*
: When present, this attribute specifies the default number of columns in this group. User agents should ignore this attribute if the current column group contains one or more COL elements. The default value of this attribute is one.

width = *length*
: This attribute specifies a default width for each column in the current column group. In addition to the standard pixel and percentage values, this attribute may take the special form "0*", which means

that the width of each column in the group should be the minimum width necessary to hold the column's contents.

Attributes defined elsewhere

- `id`, `class` (document-wide identifiers)
- `lang` (language information), `dir` (text direction)
- `title` (element titles)
- `style` (inline style information)
- `onclick`, `ondblclick`, `onmousedown`, `onmouseup`, `onmouseover`, `onmousemove`, `onmouseout`, `onkeypress`, `onkeydown`, `onkeyup` (intrinsic events)
- `align`, `char`, `charoff`, `valign` (cell alignment)

A table must contain at least one column group. In the absence of any column group definitions, a table is considered to have one column group that includes all columns in the table. The `COLGROUP` element creates an explicit column group.

The `width` attribute of the `COLGROUP` element specifies a default width for each column in a column group. The special value "0*" tells user agents to set every column in a group to its minimum width. This behavior may be overridden by the presence of a `COL` element.

The table in the following example contains two column groups. The first column group contains 10 columns and the second contains 5 columns. The default width for each column in the first column group is 50 pixels. The width of each column in the second column group will be the minimum for the column.

```
<TABLE>
<COLGROUP span="10" width="50">
<COLGROUP span="5" width="0*">
<THEAD>
<TR> ...
</TABLE>
```

The `COL` element

```
<!ELEMENT COL - O EMPTY>
<!ATTLIST COL                   -- column groups and properties --
  %attrs;                          -- %coreattrs, %i18n, %events --
  span        NUMBER     1         -- number of columns spanned by group --
  width       CDATA      #IMPLIED  -- column width specification --
  %cellhalign;                     -- horizontal alignment in cells --
  %cellvalign;                     -- vertical alignment in cells --
  >
```

Start tag: ***required****, End tag:* ***forbidden***

Attribute definitions

`width` = *length*

This attribute specifies a default width for each column in the current column group. In addition to the standard pixel and percentage values, this attribute may take the special form "0*", which means that the width of the each column in the group should be the minimum width necessary to hold the column's contents. The attribute may also have the form "i*", where "i" is an integer. This is called a *relative width*. When allotting space to rows and columns, user agents allot absolute widths first, then divide up remaining available space among relative width rows or columns. Each relative width row or column receives a portion of the space proportional to the integer preceding the "*". The value "*" is equivalent to "1*".

Attributes defined elsewhere

- `id`, `class` (document-wide identifiers)
- `lang` (language information), `dir` (text direction)
- `title` (element titles)
- `style` (inline style information)
- `onclick`, `ondblclick`, `onmousedown`, `onmouseup`, `onmouseover`, `onmousemove`, `onmouseout`, `onkeypress`, `onkeydown`, `onkeyup` (intrinsic events)
- `align`, `char`, `charoff`, `valign` (cell alignment)

Each column group defined by `COLGROUP` may contain zero or more `COL` elements. A `COL` element does not define a column group in the same sense as `COLGROUP`; it is simply a way to share attribute values among columns in a column group. Note that `COL` elements are empty; they are only affected by attributes.

The `span` attribute for `COL` means the following:

- In the absence of a `span` declaration, each `COL` element represents one column.
- If the `span` attribute is set to N > 0, the current `COL` element spans N columns in the table.
- If the `span` attribute is set to 0, the current `COL` element spans the remaining columns in the table, including the current and final columns.

As for `COLGROUP`, the `width` attribute for `COL` affects the width of the columns subsumed by the element. If a `COL` element spans several columns then its `width` attribute specifies the width of each column in the span, not the width of the span as a whole.

The table in this example contains two column groups. The first group contains three columns, the second contains two columns. The available horizontal space will be alloted as follows: First the user agent will allot 30 pixels to the first column. Then, the minimal space required for the second column will be alloted to it. The remaining horizontal space will be divided into six equal portions. Column three will receive two of these portions, column four will receive one, and column five will receive three.

```
<TABLE>
<COLGROUP>
   <COL width="30">
   <COL width="0*">
   <COL width="2*">
<COLGROUP align="center">
```

```
   <COL width="1*">
   <COL width="3*" align="char" char=":">
<THEAD>
<TR> ...
</TABLE>
```

We have set the value of the `align` attribute in the second column group to "center". All cells in every column in this group will inherit this value, but may override it. In fact, the final `COL` does just that, by specifying that every cell in the column it governs will be aligned along the ":" character.

Table rows: The TR element

```
<!ELEMENT TR - O (TH|TD)+>
<!ATTLIST TR                    -- table row --
  %attrs;                          -- %coreattrs, %i18n, %events --
  %cellhalign;                     -- horizontal alignment in cells --
  %cellvalign;                     -- vertical alignment in cells --
  bgcolor     %Color     #IMPLIED  -- background color for row --
  >
```

*Start tag: **required**, End tag: **optional***

Attributes defined elsewhere

- `id`, `class` (document-wide identifiers)
- `lang` (language information), `dir` (text direction)
- `title` (element titles)
- `style` (inline style information)
- `onclick`, `ondblclick`, `onmousedown`, `onmouseup`, `onmouseover`, `onmousemove`, `onmouseout`, `onkeypress`, `onkeydown`, `onkeyup` (intrinsic events)
- `align`, `char`, `charoff`, `valign` (cell alignment)

The `TR` elements acts as a container for a row of table cells.

This sample table contains three rows, each begun by the `TR` element:

```
<TABLE>
<CAPTION>Cups of coffee consumed by each senator</CAPTION>
<TR> ...A header row...
<TR> ...First row of data...
<TR> ...Second row of data...
...the rest of the table...
</TABLE>
```

Table cells: The TH and TD elements

```
<!ELEMENT (TH|TD) - O %block>
<!ATTLIST (TH|TD)               -- header or data cell --
  %attrs;                          -- %coreattrs, %i18n, %events --
  axis        CDATA      #IMPLIED  -- defaults to cell content --
  axes        CDATA      #IMPLIED  -- list of axis names --
  nowrap      (nowrap)   #IMPLIED  -- suppress word wrap --
```

```
  bgcolor     %Color      #IMPLIED  -- cell background color --
  rowspan     NUMBER      1         -- number of rows spanned by cell --
  colspan     NUMBER      1         -- number of cols spanned by cell --
  %cellhalign;                      -- horizontal alignment in cells --
  %cellvalign;                      -- vertical alignment in cells --
  >
```

*Start tag: **required**, End tag: **optional***

Attribute definitions

`axis` = *cdata*
: This attribute defines an abbreviated name for a header cell. The default name for a cell is its content.

`axes` = *cdata-list*
: The value of this attribute, a comma-separated list of `axis` names, specifies the row and column headers that pertain to this cell. In the absence of this attribute, user agents may make other attempts to identify the cell's pertinent header cells.

`rowspan` = *integer*
: This attribute specifies the number of rows spanned by the current cell. The default value of this attribute is one ("1"). The value zero ("0") means that the cell spans all rows from the current row to the last row of the table.

`colspan` = *integer*
: This attribute specifies the number of columns spanned by the current cell. The default value of this attribute is one ("1"). The value zero ("0") means that the cell spans all columns from the current column to the last column of the table.

`nowrap`
: **Deprecated.**When present, this boolean attribute tells visual user agents to disable automatic text wrapping for this cell. Style sheets should be used instead of this attribute (e.g., the "white-space" attribute of [CSS1] . Note: if used carelessly, this attribute may result in excessively wide cells.

Attributes defined elsewhere

- `id`, `class` (document-wide identifiers)
- `lang` (language information), `dir` (text direction)
- `title` (element titles)
- `style` (inline style information)
- `onclick`, `ondblclick`, `onmousedown`, `onmouseup`, `onmouseover`, `onmousemove`, `onmouseout`, `onkeypress`, `onkeydown`, `onkeyup` (intrinsic events)
- `bgcolor` (background color)
- `align`, `char`, `charoff`, `valign` (cell alignment)

The `TH` element stores header information, while the `TD` element stores data. This distinction enables user agents to render header and data cells distinctly, even in the absence of style sheets.

Cells may be empty (i.e., contain no data).

The following table contains four columns of data, each headed by a column description.

```
<TABLE>
<CAPTION>Cups of coffee consumed by each senator</CAPTION>
<TR> <TH>Name <TH>Cups <TH>Type of Coffee <TH>Sugar?
<TR> <TD>T. Sexton <TD>10 <TD>Espresso <TD>No
<TR> <TD>J. Dinnen <TD>5 <TD>Decaf <TD>Yes
...the rest of the table...
</TABLE>
```

Your user agent renders the beginning of this table as follows:

Cups of coffee consumed by each senator

Name	Cups	Type of Coffee	Sugar?
T. Sexton	10	Espresso	No
J. Dinnen	5	Decaf	Yes

To help distinguish the cells of this table, we can set the `border` attribute of the `TABLE` element:

```
<TABLE border="border">
...the rest of the table...
</TABLE>
```

With a border, your user agent renders the beginning of this table as follows:

Cups of coffee consumed by each senator

Name	Cups	Type of Coffee	Sugar?
T. Sexton	10	Espresso	No
J. Dinnen	5	Decaf	Yes

Labeling cells

The `axis` and `axes` attributes provide a means for specifying cell labels.

Speech synthesizers may use these labels to identify the contents and location of each cell. Processing software might consider these labels as database field names when transferring a table's contents to a database.

In the following example table, we set the value of the `axis` attribute to be the last name of each senator. We also label the cell value as falling under the "Name" column.

```
<TABLE border="border">
<CAPTION>Cups of coffee consumed by each senator</CAPTION>
<TR> <TH>Name <TH>Cups <TH>Type of Coffee <TH>Sugar?
<TR> <TD axis="Sexton" axes="Name">T. Sexton <TD>10
<TD>Espresso <TD>No
<TR> <TD axis="Dinnen" axes="Name">J. Dinnen <TD>5 <TD>Decaf <TD>Yes
</TABLE>
```

Cells that span several rows or columns

Cells may span several rows or columns. The number of rows or columns spanned by a cell is set by the `rowspan` and `colspan` attributes for either the `TH` or `TD` elements.

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.

```
<TABLE border="border">
<CAPTION>Cups of coffee consumed by each senator</CAPTION>
<TR> <TH>Name <TH>Cups <TH>Type of Coffee <TH>Sugar?
<TR> <TD>T. Sexton <TD>10 <TD>Espresso <TD>No
<TR> <TD>J. Dinnen <TD>5 <TD>Decaf <TD>Yes
<TR> <TD>A. Soria <TD colspan="3"><em>Not available</em>
</TABLE>
```

This table might be rendered by a visual user agent as follows:

```
Cups of coffee consumed by each senator
 --------------------------------------
 |   Name   |Cups|Type of Coffee|Sugar?|
 --------------------------------------
 |T. Sexton|10  |Espresso      |No    |
 --------------------------------------
 |J. Dinnen|5   |Decaf         |Yes   |
 --------------------------------------
 |A. Soria |Not available             |
 --------------------------------------
```

This example illustrates how cell definitions that span more than one row or column affect the definition of later cells. Consider the following table definition:

```
<TABLE border="border">
<TR><TD>1 <TD rowspan="2">2 <TD>3
<TR><TD>4 <TD>6
<TR><TD>7 <TD>8 <TD>9
</TABLE>
```

This table might be rendered something like this:

```
-------------
| 1 | 2 | 3 |
----|   |----
| 4 |   | 6 |
----|---|----
| 7 | 8 | 9 |
-------------
```

Since the cell labeled "2" spans two rows, it affects the positions of the cells defined in the following rows. Note that if cell "6" had not been defined in row two, an extra empty cell would have been added by the user agent to complete the row.

Similarly, in the following table definition:

```
<TABLE border="border">
<TR><TD>1 <TD>2 <TD>3
<TR><TD colspan="2">4 <TD>6
<TR><TD>7 <TD>8 <TD>9
</TABLE>
```

cell "4" spans two columns, so cell "6" is placed in column three.

```
-------------
| 1 | 2 | 3 |
--------|----
| 4     | 6 |
--------|----
| 7 | 8 | 9 |
-------------
```

This example illustrates how one might create overlapping cells. In this table, cell "5" spans two rows and cell "7" spans two columns, so there is overlap in the cell between "7" and "9":

```
<TABLE border="border">
<TR><TD>1 <TD>2 <TD>3
<TR><TD>4 <TD rowspan="2">5 <TD>6
<TR><TD colspan="2">7 <TD>9
</TABLE>
```

This table might be rendered as follows to convey the overlap:

```
-------------
| 1 | 2 | 3 |
-------------
| 4 | 5 | 6 |
----|...|----
| 7 :   | 9 |
-------------
```

The rendering of overlapping cells is undefined. Rendering will vary between user agents.

Table formatting by visual user agents

The following description describes the HTML table attributes that tell visual user agents how to format tables. Style sheets will offer better control of visual table formatting. At the writing of this specification, [CSS1] did not offer mechanisms to control all aspects of visual table formatting.

This version of HTML includes mechanisms to control:

- horizontal and vertical alignment of cell contents,
- border styles
- and cell margins

Horizontal and vertical alignment

The following attributes may be set for different table elements (see their definitions).

```
<!-- horizontal alignment attributes for cell contents -->
<!ENTITY % cellhalign
  "align (left|center|right|justify|char) #IMPLIED
   char       CDATA      #IMPLIED  -- alignment char, e.g. char=':' --
   charoff    CDATA      #IMPLIED  -- offset for alignment char --"
  >
<!-- vertical alignment attributes for cell contents -->
<!ENTITY % cellvalign
  "valign (top|middle|bottom|baseline) #IMPLIED"
  >
```

Attribute definitions

`align` = `left|center|right|justify|char`
: This attribute specifies the alignment of data and the justification of text in a cell. Possible values:
  - `left`: Left-flush data/Left-justify text. This is the default value for table data.
  - `center`: Center data/Center-justify text. This is the default value for table headers.
  - `right`: Right-flush data/Right-justify text.
  - `justify`: Double-justify text.
  - `char`: Align text around a specific character.

`valign` = `top|middle|bottom|baseline`
: This attribute specifies the vertical position of data within a cell. Possible values:
  - `top`: Cell data is flush with the top of the cell.
  - `middle`: Cell data is centered vertically within the cell. This is the default value.
  - `bottom`: Cell data is flush with the bottom of the cell.
  - `baseline`: All cells in the same row as a cell whose `valign` attribute has this value should have their textual data positioned so that the first text line occurs on a baseline common to all cells in the row. This constraint does not apply to subsequent text lines in these cells.

`char` = *cdata*
: This attribute specifies a character within a text fragment which will act as an axis for alignment. The default value for this attribute is the decimal point character for the current language (as set by the `lang` attribute (e.g., the period (".") in English and the comma (",") in French). The value of this attribute is case-sensitive.

`charoff` = *length*
: When present, this attribute specifies the offset to the first occurrence of the alignment character on each line. If a line doesn't include the alignment character, it should be horizontally shifted to end at the alignment position.

When `charoff` is used so set the offset of an alignment character, the direction of offset is determined by the current text direction (set by the `dir` attribute). In left-to-right texts (the default), offset is from the left margin. In right-to-left texts, offset is from the right margin.

The table in this example aligns a row of currency values along a decimal point. We set the alignment character to "." explicitly.

```
<TABLE border="border">
<COLGROUP>
<COL><COL align="char" char=".">
<THEAD>
<TR><TH>Vegetable <TH>Cost per kilo
<TBODY>
<TR><TD>Lettuce        <TD>$1
<TR><TD>Silver carrots <TD>$10.50
<TR><TD>Golden turnips <TD>$100.30
</TABLE>
```

The formatted table should look something like this:

```
------------------------------
|   Vegetable  |Cost per kilo|
|--------------|-------------|
|Lettuce       |        $1   |
|--------------|-------------|
|Silver carrots|      $10.50|
|--------------|-------------|
|Golden turnips|     $100.30|
------------------------------
```

Inheritance of alignment specifications

The alignment of cell contents can be specified on a cell by cell basis, or inherited from enclosing elements, such as the row, column or the table itself.

The order of precedence (from highest to lowest) for the attributes `align`, `char`, and `charoff` is the following:

1. An alignment attribute set on an element within a cell's data (e.g., `P`).
2. An alignment attribute set on a cell (`TH` and `TD`).
3. An alignment attribute set on a column or column group (`COL` and `COLGROUP`). When a cell is part of a multi-column span, the alignment property is inherited from the cell definition at the beginning of the span.
4. An alignment attribute set on a row or row group (`TR`, `THEAD`, `TFOOT`, and `TBODY`). When a cell is part of a multi-row span, the alignment property is inherited from the cell definition at the beginning of the span.
5. An alignment attribute set on the table (`TABLE`).
6. The default alignment value.

The order of precedence (from highest to lowest) for the attribute `valign` (as well as the other inherited attributes `lang`, `dir`, and `style`) is the following:

1. An attribute set on an element within a cell's data (e.g., `P`).
2. An attribute set on a cell (`TH` and `TD`).
3. An attribute set on a row or row group (`TR`, `THEAD`, `TFOOT`, and `TBODY`). When a cell is part of a multi-row span, the attribute value is inherited from the cell definition at the beginning of the span.
4. An attribute set on a column or column group (`COL` and `COLGROUP`). When a cell is part of a multi-column span, the attribute value is inherited from the cell definition at the beginning of the span.
5. An attribute set on the table (`TABLE`).
6. The default attribute value.

Furthermore, when rendering cells, horizontal alignment is determined by columns in preference to rows, while for vertical alignment, rows are given preference over columns.

The default alignment for cells depends on the user agent. However, user agents should substitute the default attribute for the current directionality (i.e., not just "left" in all cases).

User agents that do not support the "justify" value of the `align` attribute may substitute the "left" value.

Borders and rules

The following attributes affect a table's external frame and internal rules.

Attribute definitions

`frame` = `void`|`above`|`below`|`hsides`|`lhs`|`rhs`|`vsides`|`box`|`border`

This attribute specifies which sides of the frame that surrounds a table will be visible. Possible values:

- `void:` No sides. This is the default value.
- `above:` The top side only.
- `below:` The bottom side only.
- `hsides:` The top and bottom sides only.
- `vsides:` The right and left sides only.
- `lhs:` The left-hand side only.
- `rhs:` The right-hand side only.
- `box:` All four sides.
- `border:` All four sides.

`rules` = `none`|`groups`|`rows`|`cols`|`all`

This attribute specifies which rules will appear between cells within a table. Possible values:

- `none:` No rules. This is the default value.
- `groups:` Rules will appear between row groups (see `THEAD`, `TFOOT`, and `TBODY`) and column groups (see `COLGROUP` and `COL`) only.
- `rows:` Rules will appear between rows only.
- `cols:` Rules will appear between columns only.

- `all:` Rules will appear between all rows and columns.

`border` = *cdata*
: This attributes specifies the width (in pixels only) of the frame around a table (see the Note below for more information about this attribute).

In the following table, borders five pixels thick will be rendered on the left- and right-hand sides of the table and rules should be displayed between all columns.

```
<TABLE border="5" frame="vsides" rules="cols">
<TR> <TD>1 <TD>2 <TD>3
<TR> <TD>4 <TD>5 <TD>6
<TR> <TD>7 <TD>8 <TD>9
</TABLE>
```

The following settings should be observed by user agents for backwards compatibility.

- Setting `border`="0" implies `frame`="void" and, unless otherwise specified, `rules`="none".
- Other values of `border` imply `frame`="border" and, unless otherwise specified, `rules`="all".
- The value "border" in the start tag of the `TABLE` element should be interpreted as the value of the `frame` attribute. It implies `rules`="all" and some default (non-zero) value for the `border` attribute.

Thus, for example:

```
<FRAME border="2"> <=> <FRAME border="2" frame="border" rules="all">
```

and

```
<FRAME border> <=> <FRAME frame="border" rules="all">
```

Note: The `border` attribute also defines the border behavior for the `OBJECT` and `IMG` elements, but takes different values for those elements.

Cell margins

Two attributes control spacing between and within cells.

Attribute definitions

`cellspacing` = *length*
: This attribute specifies how much space should be left between the table frame and the first or last cell border for each row or column, and between the cells in a table.

`cellpadding` = *length*
: This attribute specifies the amount of space between the border of the cell and its contents, on all sides of the contents.

In the following table, the `cellspacing` attribute specifies that cells will be separated from each other and from the table frame by twenty pixels. The `cellpadding` attribute specifies that the top margin of the cell and the bottom margin of the cell will each be separated from the cell's contents by 10% of the available vertical space (the total being 20%). Similarly, the left margin of the cell and the right margin of the cell will each be separated from the cell's contents by 10% of the available horizontal space (the total being 20%).

```
<TABLE>
<TR cellspacing="20"> <TD>Data1 <TD cellpadding="20%">Data2 <TD>Data3
</TABLE>
```

If a table or given column has a fixed width, `cellspacing` and `cellpadding` may demand more space than assigned. We recommend that user agents give these attributes precedence over the `width` attribute when a conflict occurs, but this is not a requirement.

Some sample tables

The following table samples illustrate the interaction of all the table elements.

Sample 1

In "ascii art", the following table:

```
<TABLE border="border">
<CAPTION>A test table with merged cells</CAPTION>
<TR><TH rowspan=2><TH colspan="2">Average
    <TH rowspan="2">other<BR>category<TH>Misc
<TR><TH>height<TH>weight
<TR><TH align="left">males<TD>1.9<TD>0.003
<TR><TH align="left" rowspan="2">females<TD>1.7<TD>0.002
</TABLE>
```

would be rendered something like this:

```
              A test table with merged cells
    /--------------------------------------------------\
    |          |      Average        |  other    |  Misc  |
    |          |---------------------| category  |--------|
    |          |  height  |  weight  |           |        |
    |-----------------------------------------------|--------|
    | males    | 1.9      | 0.003    |           |        |
    |-----------------------------------------------|--------|
    | females  | 1.7      | 0.002    |           |        |
    \--------------------------------------------------/
```

On your browser, the table looks like this:

A test table with merged cells

<table>
<tr><th rowspan="2"></th><th colspan="2">Average</th><th rowspan="2">other category</th><th>Misc</th></tr>
<tr><th>height</th><th>weight</th><th></th></tr>
<tr><th>males</th><td>1.9</td><td>0.003</td><td></td><td></td></tr>
<tr><th>females</th><td>1.7</td><td>0.002</td><td></td><td></td></tr>
</table>

Sample 2

This sample illustrates grouped rows and columns. The example is adapted from "Developing International Software", by Nadine Kano.

In "ascii art", the following table:

```
<TABLE border="2" frame="hsides" rules="groups">
<CAPTION>CODE-PAGE SUPPORT IN MICROSOFT WINDOWS</CAPTION>
<COLGROUP align="center">
<COLGROUP align="left">
<COLGROUP align="center" span="2">
<COLGROUP align="center" span="3">
<THEAD valign="top">
<TR>
<TH>Code-Page<br>ID
<TH>Name
<TH>ACP
<TH>OEMCP
<TH>Windows<br>NT 3.1
<TH>Windows<br>NT 3.51
<TH>Windows<br>95
<TBODY>
<TR><TD>1200<TD>Unicode (BMP of ISO/IEC-10646)<TD><TD><TD>X<TD>X<TD>*
<TR><TD>1250<TD>Windows 3.1 Eastern European<TD>X<TD><TD>X<TD>X<TD>X
<TR><TD>1251<TD>Windows 3.1 Cyrillic<TD>X<TD><TD>X<TD>X<TD>X
<TR><TD>1252<TD>Windows 3.1 US (ANSI)<TD>X<TD><TD>X<TD>X<TD>X
<TR><TD>1253<TD>Windows 3.1 Greek<TD>X<TD><TD>X<TD>X<TD>X
<TR><TD>1254<TD>Windows 3.1 Turkish<TD>X<TD><TD>X<TD>X<TD>X
<TR><TD>1255<TD>Hebrew<TD>X<TD><TD><TD><TD>X
<TR><TD>1256<TD>Arabic<TD>X<TD><TD><TD><TD>X
<TR><TD>1257<TD>Baltic<TD>X<TD><TD><TD><TD>X
<TR><TD>1361<TD>Korean (Johab)<TD>X<TD><TD><TD>**<TD>X
<TBODY>
<TR><TD>437<TD>MS-DOS United States<TD><TD>X<TD>X<TD>X<TD>X
<TR><TD>708<TD>Arabic (ASMO 708)<TD><TD>X<TD><TD><TD>X
<TR><TD>709<TD>Arabic (ASMO 449+, BCON V4)<TD><TD>X<TD><TD><TD>X
<TR><TD>710<TD>Arabic (Transparent Arabic)<TD><TD>X<TD><TD><TD>X
<TR><TD>720<TD>Arabic (Transparent ASMO)<TD><TD>X<TD><TD><TD>X
</TABLE>
```

would be rendered something like this:

CODE-PAGE SUPPORT IN MICROSOFT WINDOWS

Code-Page ID	Name	ACP	OEMCP	Windows NT 3.1	Windows NT 3.51	Windows 95
1200	Unicode (BMP of ISO 10646)			X	X	*
1250	Windows 3.1 Eastern European	X		X	X	X
1251	Windows 3.1 Cyrillic	X		X	X	X
1252	Windows 3.1 US (ANSI)	X		X	X	X
1253	Windows 3.1 Greek	X		X	X	X
1254	Windows 3.1 Turkish	X		X	X	X
1255	Hebrew	X				X
1256	Arabic	X				X
1257	Baltic	X				X
1361	Korean (Johab)	X			**	X
437	MS-DOS United States		X	X	X	X
708	Arabic (ASMO 708)		X			X
709	Arabic (ASMO 449+, BCON V4)		X			X
710	Arabic (Transparent Arabic)		X			X
720	Arabic (Transparent ASMO)		X			X

On your user agent, this tables is rendered as follows:

CODE-PAGE SUPPORT IN MICROSOFT WINDOWS

Code-Page ID	Name	ACP	OEMCP	Windows NT 3.1	Windows NT 3.51	Windows 95
1200	Unicode (BMP of ISO/IEC-10646)			X	X	*
1250	Windows 3.1 Eastern European	X		X	X	X
1251	Windows 3.1 Cyrillic	X		X	X	X
1252	Windows 3.1 US (ANSI)	X		X	X	X
1253	Windows 3.1 Greek	X		X	X	X
1254	Windows 3.1 Turkish	X		X	X	X
1255	Hebrew	X				X
1256	Arabic	X				X
1257	Baltic	X				X
1361	Korean (Johab)	X			**	X
437	MS-DOS United States		X	X	X	X
708	Arabic (ASMO 708)		X			X
709	Arabic (ASMO 449+, BCON V4)		X			X
710	Arabic (Transparent Arabic)		X			X
720	Arabic (Transparent ASMO)		X			X

This example illustrates how `COLGROUP` can be used to group columns and set the default column alignment. Similarly, `TBODY` is used to group rows. The `frame` and `rules` attributes tell the user agent which borders and rules to render.

Links

Contents

Up to now, the specification has dealt with HTML constructs that add structure to a single document. In this section, we introduce concepts and constructs that allow authors to create links between documents.

Path information: the BASE element

```
<!ELEMENT BASE - O EMPTY>
<!ATTLIST BASE
  href        %URL       #REQUIRED
  target      CDATA      #IMPLIED  -- where to render linked resource --
  >
```

*Start tag: **required**, End tag: **forbidden***

Attribute definitions

`href` = *url*
: This attribute specifies an absolute URL that acts as the base URL for resolving relative URLs.

Attributes defined elsewhere

- `target` (target frame information)

It is important to consider the issue of paths when linking to another document or including an object. In HTML, path information is always specified by a URL. Relative URLs are resolved according to a base URL, which may come from a variety of sources (see the section on relative URLs for information about sources of base URLs). The `BASE` element allows authors to specify a document's base URL explicitly.

When present, the `BASE` element must appear in the `HEAD` section of an HTML document. The scope of the `BASE` element is the current document only.

For example, given the following `BASE` declaration:

```
<HTML>
<HEAD>
<BASE href="http://www.barre.fr/fou/intro.html">
</HEAD>
...
</HTML>
```

the relative URL "../gee/foo.html" would resolve to:

```
http://www.barre.fr/gee/foo.html
```

Links and anchors

A HTML link is a connection from one Web resource to another. Though a very simple concept, links have been one of the key reasons the Web has been so successful.

Definitions of links and anchors

An HTML link has two ends and a direction. The link starts at the "source" end and points to the "destination" end.

Every link definition specifies both the source and the destination of the link. One end is always defined as the location where the link definition occurs. The other end is specified by an attribute in the link definition.

A link end refers to some Web resource, such as an HTML document, an image, a video clip, a sound, a program, the current document, etc. A link end may also refer to an anchor. An *anchor* is a named zone within an HTML document. The zone may include text or other objects.

Uses of links

Defined this way, links have no inherent semantics; they just associate a source and a destination. However, links also include *type information* that allows user agents to interpret them in interesting ways.

Of course, by far the most common *use* of a link is to retrieve another Web resource (e.g., by clicking with a mouse, activating the link with a voice command, etc.).

Web-surfing is not the only use of links, however. For instance, authors may define links that specify the "next" and "previous" documents in a series of documents. User agents can render such links with navigation tools rather than as part of the document's contents.

Similarly, authors may use links to define a print order for a series of documents. User agents may follow these links to produce a coherent printed version of a manual or book.

Making heads and tails of links

As defined, a link has two ends (the source and the destination), one of which is specified by the location where the link definition occurs. But is this end the source or the destination of the link?

Imagine you are the author of a book written in HTML, and you want your readers to be able to read the book from beginning to end, chapter by chapter. If each chapter is in a separate HTML document, you can represent the order information with links: each document would include two links, one pointing to the previous chapter and one to the next. The link designating the previous chapter has its destination in the current document and its source in the preceding chapter. The link designating the next chapter has its source in the current document and its destination in the next chapter. By recording this structural information in the link definitions, you enable user agents to present the ordered relationships in interesting ways, such as navigation buttons, menus, etc.

The `rel` attribute specifies that the link being defined has its source in the current document. The `rev` attribute specifies that the link being defined has its destination in the current document. Furthermore, the values of these attributes give user agents some information about the type of resource located at the other end of the link. Examples illustrating their use are given below.

Elements that define links

There are two HTML elements that define links: `LINK` and `A`.

`LINK` may only appear in the `HEAD` section of an HTML document. It defines a relationship between the current document and another resource. Although `LINK` has no content, the relationships it defines may be rendered by some user agents.

The `A` element may only appear in the `BODY` of a document. It defines a relationship between a zone within the current document and another resource. `A` has content (text, images, etc.) that may be rendered with the rest of the document’s contents. User agents generally highlight this content to indicate the presence of a link.

The other important distinction between these two elements is that a link defined by `A` is generally interpreted by user agents to mean "retrieve the resource at the other end of this link". The retrieved resource may be handled by the user agent in several ways: by opening a new HTML document in the same user agent window, opening a new HTML document in a different window, starting a new program to handle the resource, etc.

The `title` attribute may be set for these elements to add information about the nature of a link. This information may be spoken by a user agent, rendered as a tool tip, cause a change in cursor image, etc.

Elements that define anchors

There are two ways to specify anchors in an HTML document:

- The `A` element.
- The `id` attribute of any element.

Links and anchors together. *The* `A` *element is used to define both links and anchors. It is possible, and even economical, to use the same* `A` *element to define a link and an anchor.*

Internationalization and links

Since links may point to documents written in different languages (possibly with different writing order) and using different character encodings, the `A` and `LINK` elements support the `lang` (language), `dir` (writing direction), and `charset` (character encoding) attributes. These attributes allow authors to advise user agents about the nature of the data at the other end of the link.

Armed with this additional knowledge, user agents should be able to avoid presenting "garbage" to the user. Instead, they may either locate resources necessary for the correct presentation of the document or, if they cannot locate the resources, they should at least warn the user that the document will be unreadable and explain the cause.

The `A` element

```
<!ELEMENT A - - (%inline)* -(A)>
<!ATTLIST A
  %attrs;                            -- %coreattrs, %i18n, %events --
  charset     CDATA      #IMPLIED  -- char encoding of linked resource --
  name        CDATA      #IMPLIED  -- named link end --
  href        %URL       #IMPLIED  -- URL for linked resource --
  target      CDATA      #IMPLIED  -- where to render resource --
  rel         CDATA      #IMPLIED  -- forward link types --
  rev         CDATA      #IMPLIED  -- reverse link types --
  accesskey   CDATA      #IMPLIED  -- accessibility key character --
  shape       %Shape     rect      -- for use with OBJECT SHAPES --
  coords      %Coords    #IMPLIED  -- for use with OBJECT SHAPES --
  tabindex    NUMBER     #IMPLIED  -- position in tabbing order --
  >
```

Start tag: ***required****, End tag:* ***required***

Attribute definitions

`name` = *cdata*
: This attribute indicates that the element is being used to define an anchor. The value of this attribute is a unique anchor name. The scope of this name is the current document. Note that this attribute shares the same name space as the `id` attribute.

`href` = *url*
: This attribute indicates that the element is being used to define a link. The value of this attribute is the location of one end of the link (the other end being defined by the location of this element).

`rel` = *cdata*
: This attribute indicates that the source of the link being defined is at the current location. The value of `href` in this case designates the destination of the link. The value of `rel` specifies the type of the link. This attribute may take a space-separated list of link types.

`rev` = *cdata*
: This attribute indicates that the destination of the link being defined is at the current location. The value of `href` in this case designates the source of the link. The value of `rev` specifies the type of

the link. This attribute may take a space-separated list of link types.

`charset` = *cdata*
: This attribute specifies the character encoding of the data designated by the link. The value of this attribute must be the name of a "charset" as defined in [RFC2045] (e.g., "euc-jp"). The default value for this attribute is "ISO-8859-1".

Attributes defined elsewhere

- `id`, `class` (document-wide identifiers)
- `lang` (language information), `dir` (text direction)
- `title` (element titles)
- `style` (inline style information)
- `onclick`, `ondblclick`, `onmousedown`, `onmouseup`, `onmouseover`, `onmousemove`, `onmouseout`, `onkeypress`, `onkeydown`, `onkeyup` (intrinsic events)
- `target` (target frame information)
- `tabindex` (tabbing navigation)
- `accesskey` (access keys)
- `shape`, `coords` (image maps)

The `A` element may define an anchor, a link, or both.

This example illustrates the definition of a link.

```
For more information about W3C, please consult the
<A href="http://www.w3.org/">W3C Web site</A>
```

This link designates the home page of the World Wide Web Consortium. When a user activates this link in a user agent, the user agent will retrieve the resource, in this case, an HTML document.

User agents generally render links in such a way as to make them obvious to users (underlining, reverse video, etc.). Rendering depends on the user agent. Rendering may vary according to whether the user has already visited the link or not. One possible rendering of the previous link might be:

```
For more information about W3C, please consult the W3C Web site.
                                                   ~~~~~~~~~~~~
```

To tell user agents explicitly what the character encoding of the destination page is, set the `charset` attribute:

```
For more information about W3C, please consult the
<A href="http://www.w3.org/" charset="ISO-8859-1">W3C Web site</A>
```

The following example illustrates the definition of an anchor. Suppose we define an anchor named "anchor-one" in the file "one.html".

```
...text before the anchor...
<A name="anchor-one">This is the location of anchor one.</A>
...text after the anchor...
```

This definition assigns an anchor to the entire document zone that contains the text "This is the location of anchor one". Usually, the contents of `A` are not rendered in any special way when `A` defines an anchor only.

Having defined the anchor, we may link to it from the same or another document. URLs that designate anchors end with "#" followed by the anchor name. Here are some examples of such URLs:

- An absolute URL: `http://www.mycompany.com/one.html#anchor-one`
- A relative URL: `../one.html#anchor-one`
- When the link is defined in the same document: `#anchor-one`

Thus, a link defined in the file "two.html" in the same directory as "one.html" would refer to the anchor as follows:

```
...text before the link...
For more information, please consult <A href="./one.html#anchor-one"> anchor one</A>.
...text after the link...
```

The `A` element in the following example specifies an anchor and a link simultaneously:

```
I just returned from vacation! Here's a
<A name="anchor-two"
   href="http://www.somecompany.com/People/Ian/vacation/family.png">
photo of my family at the lake.</A>.
```

This example contains a link to a different type of Web resource (a PNG image). Activating the link should cause the image resource to be retrieved from the Web (and possibly displayed if the system has been configured to do so).

Note: Some user agents fail to find anchors represented by empty `A` elements. For example, some user agents may not find the "empty-anchor" in the following HTML fragment:

```
<A name="empty-anchor"></A>
<EM>...some HTML...</EM>
<A href="#empty-anchor">Link to empty anchor</A>
```

Syntax of link attribute values

Values of the `name`, `rel`, and `rev` attributes observe the following:

- **Case sensitivity** Values are case-insensitive.
- **String matching** Characters with several possible representations in [ISO10646] (e.g., both precomposed and bsae+diacritic forms) match in two strings only if they have the same representation, except for case differences, in both strings. Case folding must be performed as specified in [UNICODE]. In particular, it is recommended that case-insensitive matching be performed by folding to uppercase letters from lowercase, not vice versa.

Mailto links

Authors may create links that do not lead to another document but instead cause email to be sent to an email address. When the link is activated, user agents should cause a mail program to open that includes the destination email address in the "To:" field.

To cause email to be sent when a link is activated, specify a MAILTO URL as the value of the `href` attribute.

In this example, when the user activates the link, the user agent should open a mail program with the address "joe@somplace.com" in the "To:" field.

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

Nested links

Links and anchors defined by the `A` element may not be nested.

ILLEGAL EXAMPLE:
The following example illustrates nested links. Nested links are not permitted.

```
This text contains
<A name="outer-anchor" href="next-outer.html">an outer anchor and
and link and <A name="inner-anchor" href="next-inner.html">an inner
anchor and link.</A></A>
```

Anchors with the `id` attribute

The `id` attribute may be used to position an anchor at the start tag of any element.

This example illustrates the use of the `id` attribute to position an anchor in an `H2` element. The anchor is linked to via the `A` element.

```
You may read more about this in <A href="#section2">Section Two</A>.
...later in the document
<H2 id="section2">Section Two</H2>
...later in the document
Please refer to <A href="#section2">Section Two</A> above
for more details.
```

The `id` and `name` attributes share the same name space (see [ISO10646]). This means that they cannot both define an anchor with the same name in the same document.

ILLEGAL EXAMPLE:
The following excerpt is illegal HTML since these attributes declare the same name twice in the same document.

```
<BODY>
<A href="#a1">...</A>
...
<H1 id="a1">
...pages and pages...
<A name="a1"></A>
</BODY>
```

Because of its specification in the HTML DTD, the `name` attribute may contain entities. Thus, the value `Dürst` is a valid name. The `id` attribute, on the other hand, may not contain entities.

The `LINK` element

```
<!ELEMENT LINK - O EMPTY>
<!ATTLIST LINK
  %attrs;                          -- %coreattrs, %i18n, %events --
  href        %URL       #IMPLIED  -- URL for linked resource --
  rel         CDATA      #IMPLIED  -- forward link types --
  rev         CDATA      #IMPLIED  -- reverse link types --
  type    %ContentType  #IMPLIED  -- advisory Internet content type --
  media       CDATA      #IMPLIED  -- for rendering on these media --
  target      CDATA      #IMPLIED  -- where to render linked resource --
  >
```

*Start tag: **required**, End tag: **forbidden***

Attributes defined elsewhere

- `id`, `class` (document-wide identifiers)
- `lang` (language information), `dir` (text direction)
- `title` (element titles)
- `style` (inline style information)
- `onclick`, `ondblclick`, `onmousedown`, `onmouseup`, `onmouseover`, `onmousemove`, `onmouseout`, `onkeypress`, `onkeydown`, `onkeyup` (intrinsic events)
- `href`, `rel`, `rev` (links and anchors)
- `target` (target frame information)
- `type`, `media` (header style information)

This element, which must appear in the `HEAD` section of a document (any number of times), defines a media-independent link. Although `LINK` has no content, it conveys relationship information that may be rendered by some user agents in a variety of ways (e.g., a toolbar with a drop-down menu of links).

This example illustrates how several `LINK` definitions may appear in the `HEAD` section of a document. The `rel` and `rev` attributes specify where the source and destination are for each link. The values "Index", "Next", and "Previous" are explained in the section on link types.

```
<HTML>
<HEAD>
<LINK rel="Index"    href="../index.html">
<LINK rel="Next"     href="Chapter_3.html">
<LINK rev="Previous" href="Chapter_1.html">
</HEAD>
...the rest of the document...
```

When should I use `rel` and `rev`? *It is not always necessary to identify which end is the destination and which is the source of a link. For the `A` element, you are not required to specify the `rel` and `rev` attributes. For the `LINK` element, you must choose one or the other. See the descriptions above on the distinction between `rel` and `rev`.*

Link types

The `rel` and `rev` attributes specify which end of a link definition is the destination and which is the source. In both cases, the value or values of the attribute describe the nature of the link. Both attributes may be specified in the same element start tag.

Authors may use the following recognized link types, listed here with their conventional interpretations.

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. Or, when the link type is "Next", user agents may preload the next document to save access time.

Contents
: The link refers to a document serving as a table of contents.

Index
: The link refers to a document providing an index for the current document.

Glossary
: The link refers to a document providing a glossary of terms that pertain to the current document.

Copyright
: The link refers to a copyright statement for the current document.

Next
: The link refers to the next document in an ordered series of documents. This value is generally used with `rel`.

Previous
: The link refers to the previous document in an ordered series of documents. This value is generally used with `rev`.

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

Help
: The link refers to a document offering help (more information, links to other sources information, etc.)

Bookmark
: The link refers to a bookmark. A bookmark is a link to a key entry point within an extended document. The `title` attribute may used, for example, to label the bookmark. Note that several

bookmarks may be defined in each document.

Stylesheet
: The link refers to an external style sheet. See the section on external style sheets for details.

Alternate
: The link refers to different versions of the same document. When used in tandem with the `lang` attribute, implies a translated version of the same document. When used in tandem with the `media` attribute, implies a version for a different medium.

Links and external style sheets

When the `LINK` element links an external style sheet to a document, the `type` attribute specifies the style sheet language and the `media` attribute specifies the destination medium or media. User agents may save time by retrieving from the network only those style sheets that apply to the current device.

Media types are further discussed in the section on style sheets.

Links and search engines

Authors may use the `LINK` element to provide a variety of information to search engines, including:

- Links to versions of a document written in another language.
- Links to versions of a document for different media, such as printable versions.
- Links to the front page of a collection of documents.

The examples below illustrate how language information, media types, and link types may be combined to improve document handling by search engines.

In the following example, we tell search engines where to find Dutch, Portuguese, and Arabic versions of a document.

```
<HEAD>
<LINK lang="nl" title="The manual in Dutch"
      rel="alternate"
      href="http://someplace.com/manual/dutch.html">
<LINK lang="pt" title="The manual in Portuguese"
      rel="alternate"
      href="http://someplace.com/manual/portuguese.html">
<LINK lang="ar" title="The manual in Arabic"
      dir="rtl"
      rel="alternate"
      href="http://someplace.com/manual/arabic.html">
</HEAD>
```

In the following example, we tell search engines where to find the printed version of a manual.

```
<HEAD>
<LINK media="print" title="The manual in postscript"
      rel="alternate"
      href="http://someplace.com/manual/postscript.ps">
</HEAD>
```

In the following example, we tell search engines where to find the front page of a collection of documents.

```
<HEAD>
<LINK rel="Start" title="The first page of the manual"
      href="http://someplace.com/manual/postscript.ps">
</HEAD>
```

Inclusions

Contents

The following sections describe the various mechanisms offered by HTML to include a resource in a document. HTML allows authors to include objects, images, applets, files, and image maps.

Including an object: the `OBJECT` element

```
<!ENTITY % OAlign "(texttop|middle|textmiddle|baseline|
                             textbottom|left|center|right)">

<!ELEMENT OBJECT - - (PARAM | %block)*>
<!ATTLIST OBJECT
  %attrs                          -- %coreattrs, %i18n, %events --
  declare     (declare)  #IMPLIED  -- declare but don't instantiate flag --
  classid     %URL       #IMPLIED  -- identifies an implementation --
  codebase    %URL       #IMPLIED  -- some systems need an additional URL --
  data        %URL       #IMPLIED  -- reference to object's data --
  type     %ContentType #IMPLIED  -- Internet content type for data --
  codetype %ContentType #IMPLIED  -- Internet content type for code --
  standby     CDATA      #IMPLIED  -- message to show while loading --
  align       %OAlign    #IMPLIED  -- positioning inside document --
  height      %Length    #IMPLIED  -- suggested height --
  width       %Length    #IMPLIED  -- suggested width --
  border      %Length    #IMPLIED  -- suggested link border width --
  hspace      %Length    #IMPLIED  -- suggested horizontal gutter --
  vspace      %Length    #IMPLIED  -- suggested vertical gutter --
  usemap      %URL       #IMPLIED  -- reference to image map --
  shapes      (shapes)   #IMPLIED  -- object has shaped hypertext links --
  name        %URL       #IMPLIED  -- submit as part of form --
  tabindex    NUMBER     #IMPLIED  -- position in tabbing order --
  >
```

Start tag: ***required****, End tag:* ***required***

Attribute definitions

`codebase` = *url*
: This attribute specifies the base path used to resolve relative URLs specified by `classid` (i.e., it gives the base URL when the object requires code). If this attribute is not specified, its default value is the base URL of the current document. Not all rendering mechanisms require this attribute.

`classid` = *url*
: This attribute specifies the location of a rendering mechanism via a URL.

`codetype` = *cdata*
: This attribute specifies the Internet Media Type (see [MIMETYPES]) of data expected by the rendering mechanism specified by `classid`. This attribute is optional but recommended when `classid` is specified since it allows the user agent to avoid loading information for unsupported media types. If no explicit value is given for this attribute, it defaults to the value of the `type` attribute.

`data` = *url*
: This attribute specifies the location of the data to be rendered.

`type` = *cdata*
: This attribute specifies the Internet Media Type (see [MIMETYPES]) for the data specified by `data`. This attribute is optional but recommended when `data` is specified since it allows the user agent to avoid loading information for unsupported media types. If no explicit value is given for this attribute, the user agent should attempt to determine the type of the data to be rendered.

`declare`
: When present, this boolean attribute makes the current `OBJECT` definition a declaration only. The object must be instantiated by a subsequent `OBJECT` definition referring to this declaration.

`standby` = *cdata*
: This attribute specifies a message that a user agent may render while loading the object's implementation and data.

`align` = `texttop`|`middle`|`textmiddle`|`baseline`|`textbottom`|`left`|`center`|`right`
: **Deprecated.** This attribute specifies the position of the object with respect its surrounding context. Its possible values are explained in the section on object alignment.

Attributes defined elsewhere

- `id`, `class` (document-wide identifiers)
- `lang` (language information), `dir` (text direction)
- `title` (element titles)
- `style` (inline style information)
- `onclick`, `ondblclick`, `onmousedown`, `onmouseup`, `onmouseover`, `onmousemove`, `onmouseout`, `onkeypress`, `onkeydown`, `onkeyup` (intrinsic events)
- `tabindex` (tabbing navigation)
- `shapes`, `usemap` (client side image maps)
- `name` (submitting objects with forms)
- `height`, `width`, `border`, `hspace`, `vspace` (visual presentation of objects, images, and applets)

Most user agents contain mechanisms for rendering common data types such as text, GIF images, colors, fonts, and a handful of graphic elements. To render data types they don't support natively, user agents generally run external applications. The `OBJECT` element allows authors to control whether included objects are handled by user agents internally or externally.

In the most general case, an inserted rendering mechanism specifies three types of information:

- The rendering mechanism's implementation
- The data to be rendered
- Additional values required by the rendering mechanism at run-time.

In certain cases, it may not be necessary to specify all of this information. For example, some rendering mechanisms may not require data (e.g., a self-contained applet that peforms a small animation). Other rendering mechanisms may not require run-time initialization. Finally, some rendering mechanisms may not require additional implementation information, i.e., the user agent itself may already know how to render that type of data (e.g., GIF images).

In HTML, the `OBJECT` element specifies the location of a rendering mechanism and the location of data required by the rendering mechanism. This information is specified by the attributes of the `OBJECT` element. The `PARAM` element specifies a set of run-time values. We discuss this element below, in the section on object initialization.

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

1. The user agent must first try to render the mechanism specified by the element's attribute.
2. If the user agent is not able to render this mechanism for whatever reason (configured not to, lack of resources, wrong architecture, etc.), it must try to render the element's contents.

In the following example, we insert a fictitious rendering mechanism written in the Python language that displays an analog clock. This applet requires no additional data or run-time values. The `classid` attribute specifies the location of the applet:

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

We recommend completing this declaration by including alternate text in the contents of `OBJECT`, in case the user agent cannot render the clock.

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

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

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 an alternate rendering mechanism. 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. User agents 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.

```
<OBJECT title="The Earth as seen from space"
           classid="http://www.observer.mars/TheEarth.py">
 <OBJECT data="TheEarth.mpeg" type="application/mpeg">
  <OBJECT src="TheEarth.gif">
   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 a rendering mechanism, relies on the user agent to handle the animation. We also set the `type` attribute so that a user agent that cannot render MPEG can elect not 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.*

Object initialization: the `PARAM` element

```
<!ELEMENT PARAM - O EMPTY       -- named property value -->
<!ATTLIST PARAM
  name        CDATA      #REQUIRED -- property name --
  value       CDATA      #IMPLIED  -- property value --
  valuetype (DATA|REF|OBJECT) DATA -- How to interpret value --
  type        CDATA      #IMPLIED  -- Internet media type --
  >
```

Start tag: ***required****, End tag:* ***forbidden***

Attribute definitions

`name` = *cdata*
: This attribute defines the name of a run-time parameter name, assumed to be known by the inserted object. Whether the property name is case-sensitive depends on the object.

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

`valuetype` = `data`|`ref`|`object`

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

- `data`: The value specified by `value` will be passed directly to the rendering mechanism as a string, after resolving any embedded character or numeric character entities. This is the default value for this attribute and may appear alone in the start tag of the element.
- `ref`: The value specified by `value` is a URL that designates a resource where run-time values are stored. The URL must be passed to the rendering mechanism **as is**, i.e., unresolved.
- `object`: The value specified by `value` is a fragment URL that designates an `OBJECT` declaration in the same document. In this case, the `OBJECT` definition must be identifiable by its `id` attribute.

`type` = *cdata*

This attribute specifies the Internet Media Type (see [MIMETYPES]) 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 URL designated by `value`.

The `PARAM` element specifies a set of values that may be required by a rendering mechanism at run-time. Any number of of `PARAM` elements may appear at the beginning of an `OBJECT` declaration. The syntax of names and values is assumed to be understood by the rendering mechanism. Names and values are passed to the rendering mechanism on its standard input.

We return to our clock example. This time, we suppose that the rendering mechanism (the clock) is able to handle two run-time parameters that define its initial height and width. We set the initial dimensions to 40x40 pixels with two `PARAM` elements.

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

Since the default `valuetype` for a `PARAM` element is "data", we could replace the above declarations with either:

```
<PARAM name="height" value="40">
<PARAM name="width" value="40" >
```

or:

```
<PARAM name="height" value="40" data>
<PARAM name="width" value="40"  data>
```

(The latter form -- the value "data" only for `valuetype` attribute -- is possible due to the DTD definition of this attribute.)

In the following example, run-time data for the rendering mechanism'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 URL designating the resource.

```
<OBJECT classid="http://www.gifstuff.com/gifappli"
       standby="Loading Elvis...">
<PARAM name="Init_values"
       value="./images/elvis.gif">
       valuetype="ref">
</OBJECT>
```

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

Rendering mechanisms are located by URLs. As we discussed in the section on URLs, the first segment of an absolute URL specifies the protocol used to transfer the data designated by the URL. For HTML documents, this protocol is generally "http". Some rendering mechanisms might employ other protocols. For instance, when specifying a Java rendering mechanism, you may use URLs that begin with "java" and for ActiveX applets, you may use "clsid".

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

```
<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 codetype="application/octet-stream"
           classid="java:program.start">
</OBJECT>
```

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

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

The following example specifies (with the `classid` attribute) an ActiveX rendering mechanism via a URL that begins with the protocol information "clsid". The `data` attribute locates the data to render (another clock).

```
<OBJECT classid="clsid:663C8FEF-1EF9-11CF-A3DB-080036F12502"
           data="http://www.acme.com/ole/clock.stm">
This application is not supported.
</OBJECT>
```

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 different location in the document, for example, by activating a link.
- It is possible to specify objects as run-time data for other objects.

To declare an rendering mechanism 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, you 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 rendering mechanism defined with the `declare` attribute is instantiated every time the `OBJECT` is referenced thereafter.

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

```
<OBJECT declare
           id="earth_declaration"
           data="TheEarth.mpeg"
           type="application/mpeg">
  <OBJECT src="TheEarth.gif">
   The <STRONG>Earth</STRONG> as seen from space.
  </OBJECT>
</OBJECT>
...later in the document...
Click to see 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 rendering mechanism 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`).

```
<OBJECT declare
     id="tribune"
     type="application/x-webfont"
     data="tribune.gif">
</OBJECT>
...view the poem in KublaKhan.txt here...
<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.

Object alignment

The `align` attribute has been deprecated for this element in favor of style sheets.

The `align` attribute specifies the position of an object with respect to its context.

The following values place an object in the heart of text:

- `texttop`: means the top of the object should be vertically aligned with the top of the current text line.
- `middle`: means the middle of the object should be vertically aligned with the current baseline.
- `textmiddle`: means the middle of the object should be vertically aligned with the position midway between the baseline and the x-height for the current font. The x-height is defined as the top of a lower case x in Western writing systems. If the text font is an all-caps style, user agents should use the height of a capital X. For other writing systems, user agents should align the middle of the object with the middle of the text.
- `baseline`: means the bottom of the object should be vertically aligned with the current baseline.
- `textbottom`: means the bottom of the object should be vertically aligned with the bottom of the current text line.

Three other values, `left`, `center`, and `right`, cause an object to float. They are discussed in the section on floating objects.

Including an image: the `IMG` element

```
<!-- To avoid problems with text-only UAs you need to provide
     a description with ALT, and avoid server-side image maps -->
<!ELEMENT IMG - O EMPTY      --  Embedded image -->
<!ATTLIST IMG
  %attrs;                        -- %coreattrs, %i18n, %events --
  src         %URL       #REQUIRED -- URL of image to embed --
  alt         CDATA      #IMPLIED  -- description for text only browsers --
  align       %IAlign    #IMPLIED  -- vertical or horizontal alignment --
  height      %Pixels    #IMPLIED  -- suggested height in pixels --
  width       %Pixels    #IMPLIED  -- suggested width in pixels --
  border      %Pixels    #IMPLIED  -- suggested link border width --
  hspace      %Pixels    #IMPLIED  -- suggested horizontal gutter --
  vspace      %Pixels    #IMPLIED  -- suggested vertical gutter --
  usemap      %URL       #IMPLIED  -- use client-side image map --
  ismap       (ismap)    #IMPLIED  -- use server-side image map --
  >
```

Start tag: ***required****, End tag:* ***forbidden***

Attribute definitions

`src` = *url*

This attribute specifies the location of the image resource. Examples of widely recognized image formats include GIF, JPEG, and PNG.

`align` = `bottom`|`middle`|`top`|`left`|`right`

Deprecated. This attribute specifies the position of the image with respect its surrounding context. Its values are explained in the section on image alignment.

Attributes defined elsewhere

- `id`, `class` (document-wide identifiers)
- `alt` (alternate text)
- `lang` (language information), `dir` (text direction)
- `title` (element titles)
- `style` (inline style information)
- `onclick`, `ondblclick`, `onmousedown`, `onmouseup`, `onmouseover`, `onmousemove`, `onmouseout`, `onkeypress`, `onkeydown`, `onkeyup` (intrinsic events)
- `ismap`, `usemap` (client side image maps)
- `name` (submitting objects with forms)
- `height`, `width`, `border`, `hspace`, `vspace` (visual presentation of objects, images, and applets)

The `IMG` element embeds an image in the current document at the location of the element's definition. However, we recommend using the `OBJECT` element to insert an image into a document.

The `height` and `width` attributes of this element override the natural height and width of the source image. User agents should scale the image appropriately.

In an earlier example, we defined a link to a family photo. Here, we insert the photo directly into the current document

```
...preceding text...
I just returned from vacation! Here's a photo of my family at the lake:
<IMG src="http://www.somecompany.com/People/Ian/vacation/family.png"
     alt="A photo of my family at the lake.">
```

This may be expressed with the `OBJECT` element as follows:

```
...preceding text...
I just returned from vacation! Here's a photo of my family at the lake:
<OBJECT data="http://www.somecompany.com/People/Ian/vacation/family.png"
        type="image/png">
A photo of my family at the lake.
</OBJECT>
```

The `alt` attribute specifies alternate text when the image cannot be displayed (see below for information on how to specify alternate text).

Image alignment

The `align` attribute has been deprecated for this element in favor of style sheets.

The `align` attribute specifies the position of an object with respect to its context.

The following values place an object in the heart of text:

- `bottom:` means that the bottom of the image should be vertically aligned with the current baseline. This is the default value.
- `middle:` means that the center of the image should be vertically aligned with the current baseline.
- `top:` means that the top of the image should be vertically aligned with the top of the current text line.

Two other values, `left` and `right`, cause the image to float. They are discussed in the section on floating objects.

Differing interpretations of `align`. *Existing 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.*

Including an applet: the APPLET element

APPLET is deprecated.

```
<!ELEMENT APPLET - - (PARAM | %inline)*>
<!ATTLIST APPLET
  codebase    %URL       #IMPLIED   -- optional base URL for applet --
  archive     CDATA      #IMPLIED   -- comma separated archive list --
  code        CDATA      #IMPLIED   -- applet class file --
  object      CDATA      #IMPLIED   -- serialized applet file --
  alt         CDATA      #IMPLIED   -- description for text only browsers --
  name        CDATA      #IMPLIED   -- allows applets to find each other --
  width       %Pixels    #REQUIRED  -- suggested width in pixels --
  height      %Pixels    #REQUIRED  -- suggested height in pixels --
  align       %IAlign    #IMPLIED   -- vertical or horizontal alignment --
  hspace      %Pixels    #IMPLIED   -- suggested horizontal gutter --
  vspace      %Pixels    #IMPLIED   -- suggested vertical gutter --
  >
```

Start tag: ***required****, End tag:* ***required***

Attribute definitions

`codebase` = *url*
: This attribute specifies the base URL for the applet. If this attribute is not specified, then the base URL for the applet is the same as for the current document.

`code` = *cdata*
: This attribute specifies the name of the resource that contains the applet's compiled applet subclass. The value must be a relative URL with respect to the applet base URL.

`name` = *cdata*
: 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.

`width` = *length*
: This attribute specifies the initial width of the applet display area (not counting any windows or dialogs that the applet brings up).

`height` = *length*
: This attribute specifies the initial height of the applet display area (not counting any windows or dialogs that the applet brings up).

`align` = `bottom`|`middle`|`top`|`left`|`right`
: This attribute specifies the position of the object with respect to its surrounding context. Its values are explained in the section on image alignment.

`archive` = *cdata*
: This attribute specifies the names of one or more comma-separated archives containing classes and other resources that will be "preloaded". The classes are loaded using an instance of an AppletClassLoader with the given `codebase`. Preloading resources can significantly improve the performance of applets.

`object` = *cdata*
: This attribute gives the name of the resource that contains a serialized representation of an applet. The applet will be deserialized. The `init()` method will not be invoked; but its `start()` method will. Attributes valid when the original object was serialized are not restored. Any attributes passed to this applet instance will be available to the Applet.

Attributes defined elsewhere

- `alt` (alternate text)
- `hspace`, `vspace` (visual presentation of objects, images, and applets)

This element, supported by all Java-enabled browsers, allows designers to embed a Java applet in an HTML document. It has been deprecated 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. The content must be ignored otherwise.

The following sample Java applet:

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

may be rewritten as follows with `OBJECT`:

```
<OBJECT codetype="application/octet-stream"
        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.

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/octet-stream"
        classid="AudioItem"
        width="15" height="15">
<PARAM name="snd" value="Hello.au|Welcome.au">
Java applet that plays a welcoming sound.
</OBJECT>
```

Including HTML in another HTML document

Sometimes, rather than linking to another document, it is helpful to include the contents of an HTML document in another HTML document. We recommend using the `OBJECT` element with the `data` attribute for this purpose.

For instance, the following line will include the contents of `piece_to_include.html` at the location where the `OBJECT` definition occurs.

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

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.

Careful file inclusions. *Be careful if you attempt to include a section of an HTML document defined by an anchor. The entire document after the anchor definition will be included, and you might unwittingly include unwanted end tags (for elements such as `BODY`, `HTML`, etc.) in your document.*

The `IFRAME` element may also be used to insert an inline frame containing text in an HTML document.

Including an image map in an HTML document

An image map allows users authors to specify active regions of an image or object and assign a specific action to each region (e.g., retrieve a document, run a program, etc.)

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:

- *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 where the document is housed. The server interprets the coordinates and performs some action.
- *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.

Client-side image maps are preferred over server-side image maps. It is possible to implement client-side image maps with several elements.

Non-graphical representation of image maps. *Non-graphical user agents may render client-side image maps as sets of textual links. The textual region may be activated by keyboard input.*

Client-side image maps

The following attributes are defined for several elements (`A` and `AREA`). They allow authors to specify a set of geometrical regions and associate URLs with them.

Attribute definitions

`shape` = `default`|`rect`|`circle`|`poly`
: 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` = *length-list*
: This attribute specifies the position a 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.
  - `poly`: x1, y1, x2, y2, ..., xN, yN.

  Coordinates are relative to the top, left corner of the object. All values are lengths (they may be pixel values or percentage values).

The following attribute is defined for the `OBJECT` element.

Attribute definitions

`shapes`

When set, this boolean attribute specifies that the object being defined is an image map. The contents of the `OBJECT` element will specify the active regions.

In the following example, we create a client-side image map for the `OBJECT` element by associating URLs with regions specified by a series of `A` elements.

```
<OBJECT data="navbar.gif" shapes>
 <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="circ" coords="184,200,60">Search</A> |
 <A href="top10.html" shape="poly" coords="276,0,373,28,50,50,276,0">Top Ten</A>
</OBJECT>
```

If the `OBJECT` element includes a `shapes` attribute, user agents must parse the contents of the element to look for anchors.

If two or more defined regions overlap, the region defined first takes precedence (i.e., responds to user input).

Client-side image maps with `MAP` and `AREA`

The `MAP` and `AREA` elements provide an alternate mechanism for client side image maps.

```
<!ELEMENT MAP - - (AREA)*>
<!ATTLIST MAP
  %coreattrs;                     -- id, class, style, title --
  name        CDATA     #IMPLIED
  >
```

*Start tag: **required**, End tag: **required***

```
<!ELEMENT AREA - O EMPTY>
<!ATTLIST AREA
  shape       %Shape    rect      -- controls interpretation of coords --
  coords      %Coords   #IMPLIED  -- comma separated list of values --
  href        %URL      #IMPLIED  -- this region acts as hypertext link --
  target      CDATA     #IMPLIED  -- where to render linked resource --
  nohref      (nohref)  #IMPLIED  -- this region has no action --
  alt         CDATA     #REQUIRED -- description for text only browsers --
  tabindex    NUMBER    #IMPLIED  -- position in tabbing order --
  >
```

*Start tag: **required**, End tag: **forbidden***

Attribute definitions

`nohref`

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

Attributes defined elsewhere

- `id`, `class` (document-wide identifiers)
- `lang` (language information), `dir` (text direction)
- `title` (element titles)
- `style` (inline style information)
- `name` (submitting objects with forms)
- `alt` (alternate text)
- `href` (anchor reference) `target` (frame target information)
- `tabindex` (tabbing navigation)
- `accesskey` (access keys)
- `shape`, `coords` (image maps)

Several elements (`OBJECT`, `IMG`, and `INPUT`) allow the following attribute to specify an associated client-side image map.

Attribute definitions

`usemap` = *url*
: This attribute specifies the location of a map defined by `MAP` and `AREA`.

We can rewrite the previous example in terms of `MAP` and `AREA`. We still create an `OBJECT` that will insert an image. We associate the object and the image map by setting the `usemap` attribute on the `OBJECT` and the `name` attribute of the `MAP` element to the same value.

```
<OBJECT data="navbar1.gif" usemap="#map1"></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">
 <AREA href="shortcut.html"
          alt="Go"
          shape=circ
          coords="184,200,60">
 <AREA href="top10.html"
          alt="Top Ten"
          shape="poly"
          coords="276,0,373,28,50,50,276,0">
</MAP>
```

The `alt` attribute specifies alternate text for cases when the image map may not be displayed (see below for information on how to specify alternate text).

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

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 with the `IMG` element. To do so, set the boolean attribute `ismap` in the `IMG` definition. The associated map of regions must be specified with the `usemap` attribute.

When the user activates a region of the image map, the screen coordinates are sent directly to the server where the document resides. Screen coordinates are expressed as screen pixel values. For normative information about the definition of a pixel and how to scale it, please consult [CSS1].

The location clicked is passed to the server as follows. The user agent derives a new URL from the URL specified by the `href` attribute by appending '?' followed by the x and y coordinates, separated by a comma. The link is then followed using the new URL. For instance, in the previous example, if the user clicks at at the location x=10, y=27 then the derived URL is "/cgibin/navbar.map?10,27".

In the following example, the first active region defines a client-side link. The second defines a server-side link, but doesn't assign a specific shape to it (this is accomplished with the "default" value of the `shape` attribute). Since the regions of the two links overlap, the first takes precedence of the later definition. Thus, a click anywhere but in the rectangle will cause the click's coordinates to be sent to the server.

```
<OBJECT data="game.gif" shapes>
  <A href="guide.html" shape="rect" coords="0,0,118,28">
        Rules of the Game</A>
  <A href="http://www.acme.com/cgi-bin/competition"
        ismap
        shape="default">
        Guess the location</a>
</OBJECT>
```

Visual presentation of images, objects, and applets

All `IMG` and `OBJECT` attributes that concern visual alignment or presentation have been deprecated in favor of style sheets.

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. Both attributes take values of type *length*. User agents may scale objects and images to match these values if appropriate.

The `vspace` and `hspace` attributes specify the amount of white space to be inserted to the left and right (hspace) and above and below (vspace) an image or object. The default value for this attribute is not specified, but is generally a small, non-zero length. Both attributes take values of type *length*.

Images or objects that are the content of an `A` element are sometimes surrounded by a border. The `border` attribute specifies the width of this border.

How to specify alternate text

Attribute definitions

`alt` = *cdata*
: 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`) allow authors to 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.

While alternate text may be very helpful, it must be handled with care. Authors should observe the following guidelines:

- Do not specify meaningless alternate text when including images intended to *format* a page. In this case, the alternate text should be the empty string (""). In any case, authors should not use 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.

The `alt` attribute is mandatory for the `AREA` element, but optional for `IMG`, `APPLET`, and `INPUT`.

When an author does not set the `alt` attribute for the `IMG` or `APPLET` elements, user agents should supply the alternate text, calculated in the following order:

1. If the `title` has been specified, its value should be used as alternate text.
2. Otherwise, if HTTP headers provide title information when the included object is retrieved, this information should be used as alternate text.
3. Otherwise, if the included object contains text fields (e.g., GIF images contain some text fields), information extracted from the text fields should be used as alternate text. Since user agents may have to retrieve an entire object first in order to extract textual information, user agents may adopt more economical approaches (e.g., content negotiation).
4. Otherwise, in the absence of other information, user agents should use the file name (minus the extension) as alternate text.

When an author does not set the `alt` attribute for the `INPUT` element, user agents should supply the alternate text, calculated in the following order:

1. If the `title` has been specified, its value should be used as alternate text.
2. Otherwise, if the `name` has been specified, its value should be used as alternate text.
3. Otherwise (submit and reset buttons), the value of the `type` attribute should be used as alternate text.

Presentation of HTML documents

Contents

Style Sheets

Contents

Style sheets represent a major breakthrough in how Web page designers work, by expanding their ability to improve the appearence 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. These authors were used to paper media where they had full control. They learned how 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 never 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, have limited flexibilty as well as suffering from interoperability problems, and creating hardships for people with disabilities.

Style sheets bring back the ease of control over presentation, and supercede the limited range of presentation mechanisms added to HTML over the last few years. 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.

HTML 4.0 provides support for the following features:

Flexible placement of Style Information
: Placing style sheets in separate files makes them easy to reuse. Sometimes its 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], but other style sheet languages would be possible.

Cascading Style Sheets

This is the capability provided by some style sheet languages such as CSS to allow style information from several sources to be blended together. 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 you to specify documents in a media independent way. This allows people to access Web pages using a wide variety of devices and media, e.g. graphical displays for windows, macs, and X11, set-top boxes for television sets, specially adapted phones and pda based portable devices, speech-based browsers, and braille-based tactical 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.

Alternative Styles

Authors may wish to offer readers several alternative styles for viewing a document. For instance, a compact version with small fonts, and one with larger fonts for increased legibility. This specification allows you to specify such alternatives, including which one is the default. Users should be given the opportunity of selecting between these styles or switching off style sheets altogether.

Note: This specification includes more detailed information about style sheets in sections on performance issues and new media types.

Adding style to HTML

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 languages may be more suited to highly specialized needs. HTML does not depend on one specific style sheet language. For the purposes of this document, however, we will present examples that illustrate Cascading Style Sheets ([CSS1]), abbreviated CSS.

Setting the default style sheet language

The syntax of a style rule is that of the style sheet language, not HTML. Since user agents that support style sheets must parse these rules, users must declare which style sheet languages are being employed.

Use the `META` element to set the default style sheet language for a document. For example, to set the default to CSS, put the following declaration in the `HEAD` of your document:

```
<META http-equiv="Content-Style-Type" content="text/css">
```

The default style sheet language can also be set with HTTP headers. The above `META` declaration is equivalent to the HTTP header:

```
Content-Style-Type: text/css
```

If two or more `META` declarations or HTTP headers specify the default style sheet language, the last one takes precedence. HTTP headers are considered as occuring earlier than the document `HEAD` for this purpose. In the absence of an explicit declaration, the default style sheet language is assumed to be CSS. We recommend that authoring tools provide an explicit declaration.

Conforming HTML parsers must be able to distinguish HTML from style sheet rules. HTML elements and attributes define the beginning of style sheet data. The end of style sheet data is defined as the end tag open delimiter (</) immediately followed by an SGML name start character ([a-zA-Z]). All style sheet data must be provided to user agent's appropriate style sheet handler.

Inline style information

Attribute definitions

`style` = *cdata*
: This attribute specifies style information for the current element.

The `style` attribute specifies style information for a single element. The style information is specified using the default style sheet language.

This example sets color and font size information for the text in a specific paragraph.

```
<P type="text/css" style="font-size: 12pt; color: fuschia">Aren't style
sheets wonderful?
```

Note the syntax of a CSS declaration: *name : value*. Property declarations are separated by a semi-colon.

The `style` attribute is appropriate when you want to apply a particular style to an individual HTML element. If the style will be reused for several elements, you should consider using the `STYLE` element. For the best flexibility, place styles in separate style sheets.

Header style information: the `STYLE` element

```
<!ELEMENT STYLE - - CDATA      -- style info -->
<!ATTLIST STYLE
  %i18n;                         -- lang, dir, for use with title --
  type        CDATA      #REQUIRED -- Internet content type
```

```
                                  for style language --
  media       CDATA      #IMPLIED  -- designed for use with these media --
  title       CDATA      #IMPLIED  -- advisory title --
  >
```

*Start tag: **required**, End tag: **required***

Attribute definitions

`type` = *cdata*
: This attribute specifies the style sheet language of the element's contents, thus overriding the default style sheet language. The style sheet language is specified as an Internet Media Type (e.g., "text/css"). Internet Media Types are defined in [MIMETYPES].

`media` = *cdata-list*
: This attribute specifies the intended destination medium for style information. It may be a single media type or a comma-separated list. Possible media types:
    - `screen`: Output is intended for non-paged computer screens. This is the default value.
    - `print`: Output is intended for paged, opaque material and for documents on screen viewed in print preview mode.
    - `projection`: Output is intended for projectors.
    - `braille`: Output is intended for braille tactile feedback devices
    - `speech`: Output is intended for a speech synthesizer.
    - `all`: Applies to all devices.

Attributes defined elsewhere

- `lang` (language information), `dir` (text direction)

The `STYLE` element allows authors to put style sheet rules in the header 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 from non-conforming user agents.

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">
   H1.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 are particularly useful in that they do not impose any predefined presentation. Since DIV and SPAN elements define structure only, when combined with style sheets, they allow users to extend HTML indefinitely.

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 id="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>
```

Media types

HTML enables authors to design documents that do not depend on a specific presentational medium. Thus, users may browse the Web with a wide variety of user agents: graphical displays for personal computers and workstations, set-top boxes for televisions, specially adapted telephones and pda-based portable devices, speech-based browsers, and braille-based tactile devices.

The `media` attribute specifies the intended output for a style rule. By setting the `media` attribute, authors may allow user agents to avoid retrieving from the network style sheets that do not apply to a given device.

The following sample declarations all apply to the `H1` element. When displayed on a computer screen, all instances will be centered and blue. When printed, all instances will be centered. We specify a different style altogether for speech synthesizers.

```
<HEAD>
 <STYLE type="text/css" media="screen">
    H1 { color: blue}
 </STYLE>

 <STYLE type="text/css" media="screen, print">
   H1 { text-align: center }
 </STYLE>

 <STYLE type="text/acss" media="speech">
   H1 { cue-before: url(bell.aiff); cue-after: url(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.

The previous example may be rewritten to refer to external style sheets (instead of using the `STYLE` element) in conjunction with the `media` attribute. User agents may consult the `media` attribute and retrieve only those style sheets appropriate for the destination medium.

```
<HEAD>
<LINK href="doc1-screen.css" rel="stylesheet"
     type="text/css" media="screen">
<LINK href="doc1-print.css" rel="stylesheet"
     type="text/css" media="print">
<LINK href="doc1-speech.css" rel="stylesheet"
     type="text/css" media="speech">
</HEAD>
```

See the following section on external style sheets for more information.

External style sheets

Style sheets may be defined separately from an HTML document. This has the advantage of offering the ability:

- to share style sheets across a number of documents (and sites)
- to change the style sheet without modifications to the document

When style sheets are enabled, users may be offered a choice of styles. Each style is potentially a cascade of several style sheets. Some style sheets (known as *persistent*) are applied independently of the user's choice (as long as it refers to the correct media type), while others (known as *alternate*) only apply to specific choices. A *default style* sheet is one that applies when the page is loaded, but which the user can disable in favor of an *alternate style* sheet.

Use the `LINK` element to designate an external style sheet. You must set the following attributes:

- Set the value of `href` to the location of the style sheet file. The value of `href` is a URL.
- Set the value of the `rel` attribute to indicate whether the style sheet is persistent (rel="stylesheet"), default (rel="stylesheet"), or alternate (rel="alternate stylesheet").
- Set the value of the `title` attribute when the style sheet is a default style sheet, i.e., when it may be activated or deactivated by the user.

In this example, we first specify a persistent external style sheet in the file `mystyle.css`.

```
<LINK href="mystyle.css" rel="stylesheet">
```

Setting the `title` attribute changes the style sheet from persistent to default; user agents should offer users the possibility of applying named styles, based upon the `title` attribute.

```
<LINK href="mystyle.css" title="Compact" rel="stylesheet">
```

Adding the keyword "alternate" to the `rel` attribute makes this an alternate style sheet.

```
<LINK href="mystyle.css" title="Medium" rel="alternate stylesheet">
```

All alternate styles sharing the same title will be applied when the user (through the user agent) activates that style. Style sheets with different titles will not be applied in this case. However, style sheets that do not have the `title` attribute set will always apply (unless the user turns off style sheets altogether).

User agents should provide a means for users to view and pick from the list of alternative styles. We recommend that the value of the `title` attribute be used to name each choice.

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 you simply provide a sequence of `LINK` and/or `STYLE` elements. The style information is cascaded in the order the elements appear in the `HEAD`. A cascade can include style sheets applicable to different media. The user agent is then responsible to filtering out those style sheets which are inapplicable to the current situation.

In the following example, we define two alternate style sheets named "compact". If the user selects the "compact" style, both external style sheets will be applied, as well as the "common.css" style sheet, (always applied since its `title` attribute is not set). If the user selects the "big print" style, the files "bigprint.css" and "common.css" will be applied by the user agent, and the "compact" style sheets will not.

```
<LINK rel="alternate stylesheet" title="compact" href="small-base.css">
<LINK rel="alternate stylesheet" title="compact"
href="small-extras.css">
<LINK rel="alternate stylesheet" title="big print" href="bigprint.css">
<LINK rel=stylesheet href="common.css">
```

Here is an example with both `LINK` and `STYLE` elements.

```
<LINK REL=stylesheet HREF="corporate.css">
<LINK REL=stylesheet HREF="techreport.css">
<STYLE TYPE="text/css">
    p.special { color: rgb(230, 100, 180) }
</STYLE>
```

Setting the default named style

Use the `META` element to set the default named style for a document. For example, to set the default named style in a document to "compact" (see the preceding example), include the following line in the `HEAD`:

```
<META http-equiv="Default-Style" content="compact">
```

The default style can also be set 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 default style, the last one takes precedence. HTTP headers are considered as occuring earlier than the document HEAD for this purpose. In the absence of an explicit declaration, the default style is defined by the first LINK element whose title has been set and whose rel attribute has the value "stylesheet".

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, you can specify the font family for all elements within the BODY by a single rule that applies to the BODY element.

Hiding the Content of Style Elements from non-conforming 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>
```

Specifying style through HTTP headers

Sometimes its convenient to configure a Web server to specify the style sheet to be applied to a group of pages at a site. 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. Thus,

```
Link: REL=stylesheet HREF="corporate.css"
```

corresponds to:

```
<LINK rel="stylesheet" href="corporate.css">
```

You can specify several alternative 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: rel="stylesheet" title="compact" href="compact.css"
Link: rel="alternate stylesheet" title="big print" href="bigprint.css"
```

This should also work when HTML documents are being by transported via email. Some email agents can alter the ordering of [RFC822] headers. To protect against this affecting the cascading order for style sheets specified by Link headers, you 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`.

Alignment, font styles, and horizontal rules

Contents

This section of the specification discusses some HTML elements and attributes that may be used for visual formatting. Generally speaking you are recommended to use style sheets instead. An exception is when dealing with user agents that either don't support style sheets or which don't support the particular style sheet features needed. A number of HTML 4.0 elements and attributes dealing with visual presentation are deprecated and may become obsolete in future versions of HTML.

Formatting

Background color

Attribute definitions

`bgcolor` = *color*
: **Deprecated.**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 in favor of style sheets for specifying background color information.

Alignment

It is possible to align block elements on the canvas (tables, images, objects, paragraphs, etc.) with the `align` element. Although this attribute may be set for many HTML elements, its range of possible values sometimes differs from element to element.

Attribute definitions

`align` = `left`|`center`|`right`|`justify`
: **Deprecated.** This attribute specifies the horizontal alignment of its element with respect to the surrounding context. Possible values:

- `left`: Left alignment/justification. This is the default value.
- `center`: Center alignment/justification.
- `right`: Right alignment/justification.
- `justify`: Double justification.

DEPRECATED EXAMPLE:
This example centers a heading on the canvas.

```
<H1 align="center"> How to Carve Wood </H1>
```

Using cascading style sheets, for example, you could achieve the same effect as follows:

```
<HEAD>
<STYLE>
H1 { text-align: center}
</STYLE>
</HEAD>
<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 `id` attribute on the element:

```
<HEAD>
<STYLE type="text/css">
H1.wood {text-align: center}
</STYLE>
</HEAD>
<H1 id="wood"> How to Carve Wood </H1>
```

Similarly, to double justify a paragraph on the canvas with HTML's `align` attribute:

```
<P align="justify">...Lots of paragraph text...
```

which, in cascading style sheets, would be:

```
<HEAD>
<STYLE type="text/css">
P.mypar {text-align: justify}
</STYLE>
</HEAD>
<P id="mypar">...Lots of paragraph text...
```

To double justify a series of paragraphs, group them with the `DIV` element:

```
<DIV align="justify">
<P>...text in first paragraph...
<P>...text in second paragraph...
<P>...text in third paragraph...
</DIV>
```

With cascading style sheets, this would be:

```
<HEAD>
<STYLE type="text/css">
DIV.mypars {text-align: justify}
</STYLE>
</HEAD>
<DIV id="mypars">
<P>...text in first paragraph...
<P>...text in second paragraph...
<P>...text in third paragraph...
</DIV>

To justify the entire document with cascading style sheets:
<HEAD>
<STYLE type="text/css">
BODY {text-align: justify}
</STYLE>
</HEAD>
<BODY>
...the body is justified...
</BODY>
```

The `CENTER` element is exactly equivalent to specifying the `DIV` element with the `align` attribute set to "center". **The `CENTER` element is deprecated.**

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, frames, etc., floats the object to either the left or right margin. Floating objects generally begin a new line. This attribute takes the following values to float an object:

- `left:` Floats the object to the current left margin. Subsequent text flows along the image's right side.
- `center:` Floats the object in the center of the page. Subsequent text flows down the object's left side, then continues down the right side.
- `right:` Floats the object to the current right margin. Subsequent text flows along the image's left side.

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/animage.gif">
```

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`

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>
|       |  ------
*********
```

If the `clear` attribute is set to `left` or `all`, next line will appear as follows:

```
*********  -------
|       |  -------
| 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. In cascading style sheets, 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>
...
*********  -------
|       |  -------
| table |  --<BR id="mybr">
|       |
*********
-----------------
...
</BODY>
```

Fonts

The following HTML elements specify font information. Although they are not all deprecated, their use is discouraged in favor of style sheets.

Font style elements: the `TT`, `I`, `B`, `BIG`, `SMALL`, `STRIKE`, `S`, and `U` elements

```
<!ENTITY % font
 "TT | I | B | U | S | STRIKE | BIG | SMALL">
<!ELEMENT (%font|%phrase) - - (%inline)*>
<!ATTLIST (%font|%phrase)
  %attrs;                        -- %coreattrs, %i18n, %events --
  >
```

*Start tag: **required**, End tag: **required***

Attributes defined elsewhere

- `id`, `class` (document-wide identifiers)
- `lang` (language information), `dir` (text direction)
- `title` (element titles)
- `style` (inline style information)
- `onclick`, `ondblclick`, `onmousedown`, `onmouseup`, `onmouseover`, `onmousemove`, `onmouseout`, `onkeypress`, `onkeydown` `onkeyup` (intrinsic events)

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.** Render strike-through style text.
U: Deprecated. Renders underlined text.

The following sentence shows several types of text:

```
<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.
```

Your browser renders the words 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 cascading style sheets:

```
<HEAD>
<STYLE>
P.mypar {font-style: italic; color: blue}
</STYLE>
</HEAD>
<P id="mypar">...Lots of blue italic text...
```

Font style elements may be nested and they must be properly nested. Rendering of nested font style elements depends on the user agent.

Font modifier elements: `FONT` and `BASEFONT`

`FONT` and `BASEFONT` are deprecated.

```
<!ELEMENT FONT - - (%inline)*   -- local change to font -->
<!ATTLIST FONT
  size        CDATA      #IMPLIED  -- [+]nn e.g. size="+1", size=4 --
  color       CDATA      #IMPLIED  -- #RRGGBB in hex, e.g. red: "#FF0000" --
  face        CDATA      #IMPLIED  -- comma separated list of font names --
  >
```

*Start tag: **required**, End tag: **required***

```
<!ELEMENT BASEFONT - O EMPTY>
<!ATTLIST BASEFONT
  size        CDATA      #REQUIRED -- base font size for FONT elements --
  color       CDATA      #IMPLIED  -- #RRGGBB in hex, e.g. red: "#FF0000" --
  face        CDATA      #IMPLIED  -- comma separated list of font names --
  >
```

*Start tag: **required**, End tag: **forbidden***

Attribute definitions

`size` = *cdata*
: **Deprecated.** 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*
: **Deprecated.** This attribute sets the text color.

`face` = *cdata-list*
: **Deprecated.** This attribute defines a comma-separated list of font names the user agent should search for in order of preference.

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 4.

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>
```

Your user agent renders this as follows:

size=1 size=2 size=3 size=4 size=5 size=6 size=7

The following shows 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 same thing with a base font size of 6:

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.

Rules: the `HR` element

```
<!ELEMENT HR - O EMPTY>
<!ATTLIST HR
  %coreattrs;                     -- id, class, style, title --
  %events;
  align (left|right|center) #IMPLIED
  noshade     (noshade)  #IMPLIED
  size        %Pixels    #IMPLIED
  width       %Length    #IMPLIED
  >
```

Start tag: ***required****, End tag:* ***forbidden***

Attribute definitions

`noshade`
: 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` = *length*
: **Deprecated.** This attribute specifies the height of the rule. The default value for this attribute depends on the user agent.

`width` = *length*
: **Deprecated.** 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

- `align` (alignment)
- `onclick`, `ondblclick`, `onmousedown`, `onmouseup`, `onmouseover`, `onmousemove`, `onmouseout`, `onkeypress`, `onkeydown`, `onkeyup` (intrinsic events)

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">
```

Your browser renders these rules as follows:

Frames

Contents

HTML frames allow authors to present documents in multiple views. Views 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 instance, to use three frames: one for a static banner, one for a navigation menu, and one for a main view that can be scrolled though or replaced by clicking on an item in the navigation frame.

Layout of frames

An HTML document with frames has a slightly different makeup than an HTML document without frames. A standard document has one `HEAD` section and one `BODY`. A document with frames has a `HEAD`, a `FRAMESET`, and an optional `BODY`.

The `FRAMESET` section of a document specifies the layout of views in the main user agent window.

The `BODY` section that follows the `FRAMESET` declaration provides alternate content for user agents that do not support frames or are configured not to display frames. We discuss alternate content in more detail below.

Elements that might normally be placed in the `BODY` element must not appear before the first `FRAMESET`element or the `FRAMESET` will be ignored.

The `FRAMESET` element

```
<!ELEMENT FRAMESET - - ((FRAMESET|FRAME)+ & NOFRAMES?)>
<!ATTLIST FRAMESET
  -- absolute pixel values, percentages or relative scales. --
  rows        CDATA      #IMPLIED  -- if not given, default is 1 row --
  cols        CDATA      #IMPLIED  -- if not given, default is 1 column --
  onload      %Script    #IMPLIED  -- all the frames have been loaded  --
  onunload    %Script    #IMPLIED  -- all the frames have been removed --
  >
```

Start tag: ***required****, End tag:* ***required***

Attribute definitions

`rows` = *length-list*
: This attribute specifies the layout of horizontal frames. It is a comma-separated list of lengths. If not specified, the default value is 100%.

`cols` = *length-list*
: This attribute specifies the layout of vertical frames. It is a comma-separated list of lengths. If not specified, the default value is 100%.

Attributes defined elsewhere

- `onload`, `onunload` (intrinsic events)

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. 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.

These two attributes have values that are comma-separated lists of lengths. A length may be absolute (given as a number of pixels or a percentage of the screen) or a relative length, indicated by the form "i*", where "i" is an integer. When allotting space to rows and columns, user agents allot absolute lengths first, then divide up remaining space among relative length rows or columns. The value "*" is equivalent to "1*".

Views 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 "*" (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>
```

The `FRAME` element

```
<!-- reserved frame names start with "_" otherwise starts with letter -->
<!ELEMENT FRAME - O EMPTY>
<!ATTLIST FRAME
  name        CDATA      #IMPLIED  -- name of frame for targetting --
  src         %URL       #IMPLIED  -- source of frame content --
  frameborder (1|0)      1         -- request frame borders? --
  marginwidth %Pixels    #IMPLIED  -- margin widths in pixels --
  marginheight %Pixels   #IMPLIED  -- margin height in pixels --
  noresize    (noresize) #IMPLIED  -- allow users to resize frames? --
  scrolling (yes|no|auto) auto     -- scrollbar or none --
  >
```

Start tag: ***required****, End tag:* ***forbidden***

Attribute definitions

`name` = *cdata*
: This attribute assigns a name to the current frame. This name may be the target of subsequent links.

`src` = *url*
: This attribute specifies the location of the initial document to be contained in the frame.

`noresize`
: When present, this boolean attribute tells the user agent that the frame window must not be resizeable.

`scrolling` = `auto`|`yes`|`no`
: 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`
: 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` = *length*
: 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 one pixel. The default value depends on the user agent.

`marginheight` = *length*
: 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 one pixel. The default value depends on the uesr agent.

Attributes defined elsewhere

- `target` (target frame information)

The `FRAME` element defines the contents and appearance of a single view.

Setting the initial document in a frame

The `src` attribute specifies the initial document the frame will contain. It is not possible for the contents of a frame to be in the same document as the frame's definition.

The following example example HTML document:

```
<HTML>
<FRAMESET cols="33%,33%,33%">
  <FRAMESET rows="*,200">
      <FRAME src="contents_of_frame1.html">
      <FRAME src="contents_of_frame2.gif">
  </FRAMESET>
  <FRAME src="contents_of_frame3.html">
  <FRAME src="contents_of_frame4.html">
</FRAMESET>
</HTML>
```

will create a frame layout something like this:

```
--------------------------------------------
|Frame 1      |Frame 3       |Frame 4       |
|             |              |              |
|             |              |              |
|             |              |              |
|             |              |              |
|             |              |              |
|             |              |              |
|             |              |              |
-------------|               |              |
|Frame 2      |              |              |
|             |              |              |
|             |              |              |
--------------------------------------------
```

and cause the user agent to load each file into a separate view.

ILLEGAL EXAMPLE:
The following frameset definition is not legal HTML since the contents of the second frame are in the same document as the frameset.

```
<HTML>
<FRAMESET cols="50%,50%">
  <FRAME src="contents_of_frame1.html">
  <FRAME src="#anchor_in_same_document">
</FRAMESET>
<BODY>
...some text...
<H2><A name="anchor_in_same_document">Important section</A></H2>
...some text...
</BODY>
</HTML>
```

Decorating a frame

The following example illustrates the usage of the decorative `FRAME` attributes. We specify that frame 1 will allow no scroll bars. Frame 2 will leave white space around its contents (initially, an image file) and the frame will not be resizeable. No border will be drawn between frames 3 and 4. Borders will be drawn (by default) between frames 1, 2, and 3.

```
<HTML>
<FRAMESET cols="33%,33%,33%">
  <FRAMESET rows="*,200">
      <FRAME src="contents_of_frame1.html" scrolling="no">
      <FRAME src="contents_of_frame2.gif"
                marginwidth="10" marginheight="15"
                noresize>
  </FRAMESET>
  <FRAME src="contents_of_frame3.html" border="0">
  <FRAME src="contents_of_frame4.html" border="0">
</FRAMESET>
</HTML>
```

Specifying target frame information

Attribute definitions

`target` = *cdata*
: This attribute specifies the name of a target frame where a document is to be opened.

By assigning a name to a frame via the `name` attribute, authors can refer to it as the "target" of links defined by other elements. The `target` attribute may be set for elements that create links (`A`, `LINK`), image maps (`AREA`), and forms (`FORM`).

This example illustrates how targets allow the dynamic modification of a frame's contents. First we define a frameset in the document `frameset.html`, shown here:

```
<HTML>
<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".

```
<HTML>
<BODY>
...beginning of the document...
Now you may advance to
    <A href="slide2.html" target="dynamic">slide 2.</A>
...more document...
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: Once a frame's content is changed dynamically, the original frameset definition no longer reflects the true contents of each frame; the frameset definition does not change.

The is currently no way to encode the entire state of a frameset in a URL. 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.

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` attributes in each element. This is done by setting the `target` attribute of the `BASE` element.

We return to the previous example, this time factorizing the target information by defining it in the `BASE` element and removing it from the `A` elements.

```
<HTML>
<HEAD>
<BASE target="dynamic">
</HEAD>
<BODY>
...beginning of the document...
Now you may advance to <A href="slide2.html">slide 2.</A>
...more document...
You're doing great. Now on to
       <A href="slide3.html">slide 3.</A>
</BODY>
</HTML>
```

Target semantics

There are several methods for making a frame the target of a link. Here we define their interaction.

1. If an element has its target attribute set to a known frame, when the element is activated, the document designated by the element will 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 determines the frame, and loading obeys the same semantics as 1.
3. If neither the element nor the `BASE` element refer to a target, the document designated by the element will be loaded into the frame containing the element.
4. If any target refers to an unknown frame `F`, the user agent will create a new window and frame, assign the name `F` to the frame, and load the document designated by the element in the new frame.

User agents may provide users with a mechanism to override the `target` attribute.

Target names

Except for the reserved names listed below, target names 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 cancelling all other frames). This value is equivalent to `_self` if the current frame has no parent.

Alternate content

We strongly recommend providing alternate versions of content for those user agents that do not support frames or are configured not to display frames.

User agents that do not support frames must display the `BODY` section that follows the outermost `FRAMESET` of a document. User agents that do support frames must ignore this `BODY` unless currently configured not to display frames.

The `NOFRAMES` element

```
<!--
 The following is quite complicated because of the mixed
 content model. However it’s actually only meant to contain
 either BODY or %block.
 -->
<!ELEMENT NOFRAMES - -
 (#PCDATA,((BODY,#PCDATA)|
          (((%blocklevel)|%font|%phrase|%special|%formctrl),%block)))>
```

*Start tag: **required**, End tag: **required***

The `NOFRAMES` element specifies content that should be displayed only when frames are not being displayed. 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.

Suppose we have a sample frameset defined in "top.html" that designates a document ("main.html") and a special table of contents ("table_of_contents.html") related to the main document. Here is "top.html":

```
<HTML>
<FRAMESET cols="50%, 50%">
   <FRAME src="main.html">
   <FRAME src="table_of_contents.html">
</FRAMESET>
</HTML>
```

What happens when the user reads "top.html" and the user agent is not displaying frames? The user won't see anything since we have not specified alternate content in the `BODY` of "top.html". If we insert "table_of_contents.html" and "main.html" directly in the `BODY`, we solve the problem of associating the two documents, but we may cause user agents that support frames to retrieve the same data twice: one copy associated with the frameset and one copy inserted in the `BODY`.

It is more economical to include the table of contents at the top of "main.html" within a `NOFRAMES` element:

```
<!-- This is main.html -->
<HTML>
<BODY>
<NOFRAMES>
...the table of contents here...
</NOFRAMES>
...the rest of the document...
</BODY>
</HTML>
```

and to link to "main.html" from "top.html" for the case when frames are not displayed:

```
<!-- This is top.html -->
<HTML>
<FRAMESET cols="50%, 50%">
   <FRAME src="main.html">
   <FRAME src="table_of_contents.html">
</FRAMESET>
<BODY>
Click <A href="main.html">here</A> for a non-frames version.
</BODY>
</HTML>
```

Inline frames: the `IFRAME` element

```
<!ELEMENT IFRAME - - %block>
<!ATTLIST IFRAME
  name        CDATA      #IMPLIED  -- name of frame for targetting --
  src         %URL       #IMPLIED  -- source of frame content --
  frameborder (1|0)      1         -- request frame borders? --
  marginwidth %Pixels    #IMPLIED  -- margin widths in pixels --
  marginheight %Pixels   #IMPLIED  -- margin height in pixels --
  scrolling (yes|no|auto) auto     -- scrollbar or none --
  align       %IAlign    #IMPLIED  -- vertical or horizontal alignment --
  height      %Length    #IMPLIED  -- suggested height --
  width       %Length    #IMPLIED  -- suggested width --
  >
```

Start tag: ***required****, End tag:* ***required***

Attribute definitions

`width` = *length*
: The width of the inline frame.

`height` = *length*
: The height of the inline frame.

- `name`, `src`, `frameborder`, `marginwidth`, `marginheight`, `scrolling` (frame controls and decoration)
- `target` (target frame information)
- `align` (alignment)

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 rendered 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. Click to retrieve
<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 including files in HTML documents for details.

Interactive HTML documents

Contents

Forms

Contents

An HTML form is a section of a document containing normal content, markup, and special elements called *controls*. Controls respond to and accept user input. Users generally "complete" forms by entering text, selecting menu items, etc., and then submitting the form for processing. Submitted forms may either be mailed to another user or fed to a program for treatment.

Controls may be check boxes, radio buttons, labels, menus, etc. Each control may be assigned a name. When the form is submitted, some controls (depending on their state) have their name and current value submitted along with the form. The nature of the value submitted depends on the control (e.g., the value of a text box is the input text).

Note: This specification includes more detailed information about forms in sections on form display issues. Further information on encoding form contents is expected to be added in later revisions to this draft.

The FORM element

```
<!ELEMENT FORM - - %block -(FORM)>
<!ATTLIST FORM
  %attrs;                          -- %coreattrs, %i18n, %events --
  action      %URL       #REQUIRED -- server-side form handler --
  method      (GET|POST) GET       -- HTTP method used to submit the form --
  enctype     %ContentType; "application/x-www-form-urlencoded"
  onsubmit    %Script    #IMPLIED  -- the form was submitted --
```

```
  onreset      %Script    #IMPLIED  -- the form was reset --
  target       CDATA      #IMPLIED  -- where to render result --
  accept-charset CDATA    #IMPLIED  -- list of supported charsets --
  >
```

Start tag: ***required****, End tag:* ***required***

Attribute definitions

`action` = *url*
: This attribute specifies a program for handling the submitted form. It may be an HTTP URL (to submit the form to a program) or a MAILTO URL (to email the form).

`method` = `get`|`post`
: This attribute specifies which HTTP method will be used to submit name/value pairs to the form handler. The Possible values:
  - `post`**:** Use the HTTP POST method. The POST method includes name/value pairs in the body of the form and not in the URL specified by the `action` attribute.
  - `get`**: Deprecated.** Use the HTTP GET method. The GET method appends name/value pairs to the URL specified by `action` and sends this new URL to the server. This is the default value for backwards compatibility. This value has been deprecated for reasons of internationalization.

`enctype` = *cdata*
: This attribute specifies the Internet Media Type (see [MIMETYPES]) 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 when the returned document includes submitted files.

`accept-charset` = *cdata*
: This attribute specifies the list of character encodings for input data that must be accepted by the server processing this form. The value is a space and/or comma-delimited list of "charsets" as defined in [RFC2045]. The server must interpret this list as an exclusive-or list, i.e., the server must be 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` = *cdata*
: This attribute specifies a comma-separated list of MIME types that a server processing this form will handle correctly. User agents may use this information to filter out nonconformant files when prompting a user to select files to be sent to the server (cf. the `INPUT` element when `type`="file").

Attributes defined elsewhere

- `id`, `class` (document-wide identifiers)
- `lang` (language information), `dir` (text direction)
- `style` (inline style information)
- `title` (element titles)
- `target` (target frame information)
- `onsubmit`, `onreset`, (intrinsic events)
- `onclick`, `ondblclick`, `onmousedown`, `onmouseup`, `onmouseover`, `onmousemove`,

onmouseout, onkeypress, onkeydown, onkeyup (intrinsic events)

The FORM element acts as a container for controls. 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 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 to restrict the user's ability to enter unrecognized characters.

A form can contain text and markup (paragraphs, lists, etc.) as well as the controls listed below.

The scope of the name attribute for any controls within a FORM element is the FORM element.

The following example specifies that the submitted form will be processed by the "adduser" program. The form will be sent to the program using the HTTP POST method.

```
<FORM action="http://somesite.com/prog/adduser" method="post">
...form contents...
</FORM>
```

The following example shows how to send a submitted form to an email address.

```
<FORM action="mailto:Kligor.T@gee.whiz.com" method="post">
...form contents...
</FORM>
```

Controls

The following control elements generally appear within a FORM element declaration. However, these elements may also appear outside of a FORM element declaration when they are used to build user interfaces. This is discussed later in this specification, in the section on intrinsic events.

Control labels
Some form controls automatically have labels associated with them (press buttons created by INPUT and BUTTON) while most do not (text fields created by INPUT and TEXTAREA, checkboxes and radio buttons created by INPUT, and menus created by SELECT).

For those controls that have implicit labels, user agents should take the value of the value attribute for the label string.

For those controls without implicit labels, authors must provide labels before or after the control element's definition. This is illustrated in the examples below.

The `INPUT` element

```
<!ENTITY % InputType
  "(TEXT | PASSWORD | CHECKBOX |
    RADIO | SUBMIT | RESET |
    FILE | HIDDEN | IMAGE | BUTTON)"
   >

<!-- HSPACE and VSPACE missing due to lack of widespread support -->
<!ELEMENT INPUT - O EMPTY>
<!ATTLIST INPUT
  %attrs;                          -- %coreattrs, %i18n, %events --
  type      %InputType   TEXT      -- what kind of widget is needed --
  name        CDATA      #IMPLIED  -- required for all but submit & reset --
  value       CDATA      #IMPLIED  -- required for radio and checkboxes --
  checked   (checked)    #IMPLIED  -- for radio buttons and check boxes --
  disabled  (disabled)   #IMPLIED  -- control is 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         %URL       #IMPLIED  -- for fields with images --
  alt         CDATA      #IMPLIED  -- description for text only browsers --
  usemap      %URL       #IMPLIED  -- use client-side image map --
  align       %IAlign    #IMPLIED  -- vertical or horizontal alignment --
  tabindex    NUMBER     #IMPLIED  -- position in tabbing order --
  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 --
  accept      CDATA      #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`
: This attribute specifies the type of input control to create. We discuss input control types below. The default value for this attribute is "text".

`name` = *cdata*
: This attribute assigns a name to the control. This name will be paired with the current value of the control if the element’s value is submitted along with the form.

`value` = *cdata*
: This attribute specifies the initial value of the control. It is optional except when the control type is "radio".

`size` = *cdata*
: This attribute tells the user agent the initial width of the control. The width is given in pixels, except for control types "text" and "password" when it is the (integer) number of characters.

`maxlength` = *integer*
: When the control type is "text" or "password", this attribute specifies the maximum number of characters that may be entered. 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`
: When the control type is "radio", this boolean attribute specifies that the radio button is on. This attribute must be ignored for other control types.

`src` = *url*
: When the control type is "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)
- `lang` (language information), `dir` (text direction)
- `title` (element titles)
- `style` (inline style information)
- `alt` (alternate text)
- `align` (alignment)
- `accept` (legal MIME types for a server)
- `readonly` (read-only input controls)
- `disabled` (disabled input controls)
- `tabindex` (tabbing navigation)
- `usemap` (client-side image maps)
- `onfocus`, `onblur`, `onselect`, `onchange` (intrinsic events)
- `onclick`, `ondblclick`, `onmousedown`, `onmouseup`, `onmouseover`, `onmousemove`, `onmouseout`, `onkeypress`, `onkeydown`, `onkeyup` (intrinsic events)

The nature of a control defined by the `INPUT` element depends on the value of the `type` attribute.

Input types

The `INPUT` element's `type` attribute determines which control will be created.

text
: This type creates a single-line text box. The value submitted by a text control is the input text.

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 is used for sensitive input such as passwords. The value submitted by a password control is the input text (not the rendering).

checkbox
: A checkbox is an on/off switch. When the switch is on, the value of the checkbox is "active". When the switch is off, the value is inactive. The checkbox value is only submitted with the form when the switch is on.

  Several checkboxes within the same form may bear the same name. Upon submission, each "on" checkbox with the same name submits a name/value pair with the same name component. This allows users to select more than one value for a given property.

radio

A radio button is an on/off switch. When the switch is on, the value of the radio button is "active". When the switch is off, the value is inactive. The radio button value is only submitted with the form when the switch is on.

Several radio button within the same form may bear the same name. However, only one of these buttons may be "on" at any one time. All related buttons are set to "off" as soon as one is set to "on". Thus, for related radio buttons, only one name/value pair is ever submitted.

submit

Creates a submit button. When this button is activated by the user, the form is submitted to the location specified by the `action` attribute of the englobing `FORM` element.

A form may contain more than one submit button. Only the name/value pair of the activated submit button is submitted with the form.

image

Creates a graphical *submit* button. The value of the `src` attribute specifies the URL of the image that will decorate the button. Some users will be unable to see this image. We strongly recommend you provide a value for the `alt` attribute as an textual alternative for the image.

When a pointing device is used to click on the image, the form is submitted and the location passed to the server. The x value is measured in pixels from the left of the image, and the y value in pixels 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, you are recommended to consider alternative approaches:

- Use multiple submit buttons (each with their own image) in place of a single graphical submit button. You can use style sheets to control the positioning of these buttons.
- Use a client-side image map together with scripting.

A possible future extension would be to add the `usemap` attribute to `INPUT` for use as 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.

reset

Creates a reset button. When this button is activated by the user, all of the form's controls have their values reset to the initial values specified by their `value` attributes. The name/value for a reset button are not submitted with the form.

button

Creates a push button that has no default behavior. The behavior of the button is defined by associating the button with client-side scripts that are triggered when events affecting the button occur (e.g., clicking the button). The value of the `value` attribute is the label used for the button.

For example, the following declaration causes the function named `verify` to be executed when the button is clicked. The script must be defined by a `SCRIPT` element.

```
<INPUT type="button" value="Click Me" onclick="verify()">
```

Please consult the section on intrinsic events for more information about scripting and events.

hidden

Creates a element that is not rendered by the user agent. However, the element's name and value are submitted with the form.

This control type is generally used to store information between client/server exchanges that would otherwise be loss due to the stateless nature of HTTP.

`INPUT` controls of type `hidden` have their values submitted with the form. The same holds for controls that are not rendered because of style information. The following control, though hidden by the user agent, will have its value submitted with the form.

```
<INPUT type="password" style="display:none"
          name="invisible-password"
          value="mypassword">
```

file

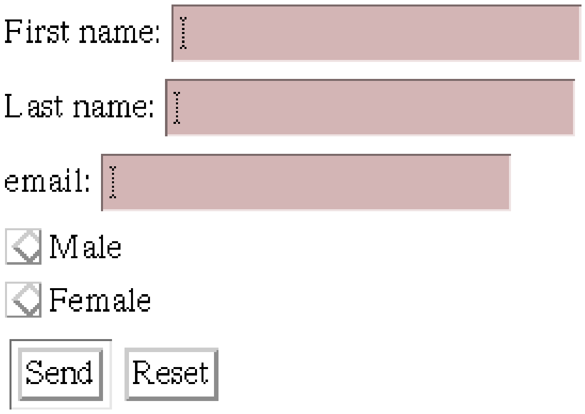
Prompts the user for a file name. When the form is submitted, the contents of the file are submitted to the server as well as other user input.

User agents should encapsulate multiple files in a MIME multipart document (see [RFC2045]). This mechanism encapsulates each file in a a body-part of a multipart MIME body that is sent as the HTTP entity. Each each body part can be labeled with an appropriate "Content-Type", including if necessary a "charset" parameter that specifies the character encoding.

The following sample HTML fragment defines a simple form that allows the user to enter a first name, last name, email address, and sex. When the submit button is activated, the form is 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">
</FORM>
```

This form might be rendered as follows:



In the section on the LABEL element, we discuss marking up labels such as "First name".

The following example shows how the contents of a user-specified file may be submitted with a form. This example is based on an example from [RFC1867].

In this example, the user is prompted to enter a name and a list of names of files whose contents should be submitted with the form. By specifying the enctype value of "multipart/form-data", each file's contents are stored in a separate section of a multipart document.

```
<FORM action="http://server.dom/cgi/handle"
    enctype="multipart/form-data"
    method="post">
 What is your name? <INPUT type="text" name="name_of_sender">
 What files are you sending? <INPUT type="file" name="name_of_files">
</FORM>
```

Please consult [RFC1867] for more information about file submissions.

The **ISINDEX** element

ISINDEX is deprecated. Users should use the INPUT element instead of this element.

```
<!ELEMENT ISINDEX - O EMPTY>
<!ATTLIST ISINDEX
  %coreattrs;                       -- id, class, style, title --
  %i18n;                            -- lang, dir --
  prompt      CDATA      #IMPLIED   -- prompt message -->
```

*Start tag: **required**, End tag: **forbidden***

Attribute definitions

prompt = *cdata*
: **Deprecated.** This attribute specifies a prompt string for the input field.

Attributes defined elsewhere

- `id`, `class` (document-wide identifiers)
- `lang` (language information), `dir` (text direction)
- `title` (element titles)
- `style` (inline style information)

The `ISINDEX` element causes the user agent to prompt the user for a single line of input (allowing any number of characters). The user agent 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: ">
```

is equivalent to the following `INPUT` declaration:

```
<FORM action="..." method="post">
Enter your search phrase: <INPUT type="text">
</FORM>
```

Semantics of* `ISINDEX`*. *Currently, the semantics for* `ISINDEX` *are only well-defined when the base URL for the enclosing document is an HTTP URL. In practice, the input string is restricted to Latin-1 as there is no mechanism for the URL to specify a different character set.*

The `BUTTON` element

```
<!ELEMENT BUTTON - -
     (%inline | %blocklevel)* -(A | %formctrl | FORM | ISINDEX | FIELDSET)>
<!ATTLIST BUTTON
  %attrs;                              -- %coreattrs, %i18n, %events --
  name        CDATA      #IMPLIED  -- for scripting/forms as submit button --
  value       CDATA      #IMPLIED  -- gets passed to server when submitted --
  type    (submit|reset) #IMPLIED  -- for use as form submit/reset button --
  disabled   (disabled)  #IMPLIED  -- control is 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 --
  >
```

Start tag: ***required****, End tag:* ***required***

Attribute definitions

`name` = *cdata*
This attribute assigns a name to the button.
`value` = *cdata*
This attribute assigns a value to the button.
`type` = `button|submit|reset`
This attribute declares the type of the button. When this attribute is not set, the button's behavior is undefined. Possible values:

- `button`: Creates a simple push button intended to trigger a script.
- `submit`: Creates a button that submits an englobing form. This is the default value.
- `reset`: Creates a button that resets an englobing form.

Attributes defined elsewhere

- `disabled` (disabled input controls)
- `tabindex` (tabbing navigation)
- `usemap` (client-side image maps)
- `onfocus`, `onblur` (intrinsic events)
- `onclick`, `ondblclick`, `onmousedown`, `onmouseup`, `onmouseover`, `onmousemove`, `onmouseout`, `onkeypress`, `onkeydown`, `onkeyup` (intrinsic events)

A `BUTTON` element whose type is "submit" is very similar to an `INPUT` element whose type is "submit". They both cause a form to be submitted, but the `BUTTON` element allows richer presentational possibilities.

A `BUTTON` element whose type is "submit" and whose content is an image (e.g., the `IMG` element) is very similar to an `INPUT` element whose type is "image". They both cause a form to be submitted, but their presentation is different. In this context, an `INPUT` element is supposed to be rendered as a "flat" image, while a `BUTTON` is supposed to be rendered as a button (e.g., with relief and an up/down motion when clicked).

The following example expands a previous example by substituting the `INPUT` elements that create submit and reset buttons with button `BUTTON` instances. 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>
</FORM>
```

If a `BUTTON` is used with an `IMG` element, you are recommended to exploit the `IMG` element’s `alt` attribute to provide a description for users unable to see the image.

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 considered legal HTML.

```
<BUTTON>
<IMG src="foo.gif" usemap="...">
</BUTTON>
```

A BUTTON element whose type is "reset" is very similar to an INPUT element whose type is "reset". They both cause controls to regain their initial values, but the BUTTON element allows richer presentation.

The BUTTON element may also be used together with scripts, in which case it's type should be "button". When such a button is activated, a client-side script is executed. We discuss this use of BUTTON later in the specification in the section on intrinsic events.

The **SELECT** and **OPTION** elements

```
<!ELEMENT SELECT - - (OPTION+)>
<!ATTLIST SELECT
  %attrs;                          -- %coreattrs, %i18n, %events --
  name        CDATA      #REQUIRED -- field name --
  size        NUMBER     #IMPLIED  -- rows visible --
  multiple  (multiple)   #IMPLIED  -- default is single selection --
  disabled  (disabled)   #IMPLIED  -- control is 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 --
  onselect    %Script    #IMPLIED  -- some text was selected --
  onchange    %Script    #IMPLIED  -- the element value was changed --
  >
```

*Start tag: **required**, End tag: **required***

SELECT Attribute definitions

name = *cdata*
: This attribute assigns a name to the element. This name will be paired with any selected values when the form is submitted.

size = *integer*
: This attribute specifies the number of rows to be rendered by the user agent. The number of rows may be smaller than the number of possible choices. In this case, the user agent should provide a scrolling mechanism for accessing all possible choices.

multiple
: When set, this boolean attribute allows multiple selections. When not set, the SELECT element only permits single selections. Traditionally, visual user agents render multiple-selection elements as list boxes, while single-selection elements are rendered as drop-down menus.

The SELECT element creates a list of choices that may be selected by the user. Each SELECT element must contain at least one choice. Each choice is specified by an instance of the OPTION element.

```
<!ELEMENT OPTION - O (#PCDATA)*>
<!ATTLIST OPTION
  %attrs;                          -- %coreattrs, %i18n, %events --
  selected  (selected)   #IMPLIED
  disabled  (disabled)   #IMPLIED  -- control is unavailable in this context --
  value       CDATA      #IMPLIED  -- defaults to element content --
  >
```

*Start tag: **required**, End tag: **optional***

OPTION Attribute definitions

`selected`
: When set, this boolean attribute specifies that this option is selected (initially or by the user).

`value` = *cdata*
: This attribute specifies the value to be submitted for this choice if the choice is selected when the form is submitted. The value is paired with the `name` assigned to the `SELECT` element. If this attribute is not set, the submitted value defaults to the content of the `OPTION` element.

Attributes defined elsewhere

- `id`, `class` (document-wide identifiers)
- `lang` (language information), `dir` (text direction)
- `title` (element titles)
- `style` (inline style information)
- `disabled` (disabled input controls)
- `tabindex` (tabbing navigation)
- `onfocus`, `onblur`, `onchange` (intrinsic events)
- `onclick`, `ondblclick`, `onmousedown`, `onmouseup`, `onmouseover`, `onmousemove`, `onmouseout`, `onkeypress`, `onkeydown`, `onkeyup` (intrinsic events)

User agents should use the content of the `OPTION` element as the displayed choice.

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 initially selected but may be deselected by the user. The remaining components are not initially 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 must be made available through a scrolling mechanism.

The `SELECT` is followed by submit and reset buttons.

```
<FORM action="http://somesite.com/prog/component-select" method="post">
   <SELECT multiple size="4" name="component-select">
      <OPTION selected value="Component_1_a">Component_1</OPTION>
      <OPTION selected value="Component_1_b">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">
</FORM>
```

When the form is submitted, each selected choice will be paired with the name "component-select" and submitted. The submitted value of each `OPTION` will be its contents, except where overridden by the `value` attribute (here, in the first two components).

The **`TEXTAREA`** element

```
<!ELEMENT TEXTAREA - - (#PCDATA)*>
<!ATTLIST TEXTAREA
  %attrs;                         -- %coreattrs, %i18n, %events --
  name        CDATA      #REQUIRED
  rows        NUMBER     #REQUIRED
  cols        NUMBER     #REQUIRED
  disabled  (disabled)   #IMPLIED  -- control is unavailable in this context --
  readonly  (readonly)   #IMPLIED
  tabindex    NUMBER     #IMPLIED  -- position in tabbing order --
  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*
: This attribute assigns a name to the element. This name will be paired with the content of the element when submitted to the server.

`rows` = *integer*
: 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 textarea field when the contents extend beyond the visible area.

`cols` = *integer*
: 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 textarea field 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` <<<<<<< forms.src (document-wide identifiers)
- `lang` (language information), `dir` (text direction)
- `title` (element titles)
- `style` (inline style information)
- `readonly` (read-only input controls)

- `disabled` (disabled input controls)
- `tabindex` (tabbing navigation)
- `onfocus`, `onblur`, `onselect`, `onchange` (intrinsic events)
- `onclick`, `ondblclick`, `onmousedown`, `onmouseup`, `onmouseover`, `onmousemove`, `onmouseout`, `onkeypress`, `onkeydown`, `onkeyup` (intrinsic events)

The `TEXTAREA` element creates a multi-line text input control (as opposed to a single-line `INPUT` control). The content of this element provides the initial text presented by the control.

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://somesite.com/prog/text-read" method="post">
   <TEXTAREA rows="20" cols="80">
   First line of initial text.
   Second line of initial text.
   </TEXTAREA>
   <INPUT type="submit" value="Send"><INPUT type="reset">
</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.

It is recommended that user agents canonicalize line endings to CR, LF (ASCII decimal 13, 10) when submitting the field's contents. The character set for submitted data should be ISO Latin-1, unless the server has previously indicated that it can support alternative character sets.

The `LABEL` element

```
<!ELEMENT LABEL - - (%inline)* -(LABEL) -- field label text -->
<!ATTLIST LABEL
  %attrs;                           -- %coreattrs, %i18n, %events --
  for         IDREF      #IMPLIED  -- matches field ID value --
  disabled    (disabled) #IMPLIED  -- control is unavailable in this context --
  accesskey   CDATA      #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` = *control-name*
: This attribute explicitly associates the label being defined with another control. The value of this attribute must be the value of the `id` attribute of some other control in the same document. In the absence of this attribute, the label being defined is associated with its contents.

Attributes defined elsewhere

- `id`, `class` (document-wide identifiers)
- `lang` (language information), `dir` (text direction)
- `title` (element titles)
- `style` (inline style information)
- `disabled` (disabled input controls)
- `accesskey` (access keys)
- `tabindex` (tabbing navigation)
- `onclick`,
- `onfocus`, `onblur` (intrinsic events)
- `onclick`, `ondblclick`, `onmousedown`, `onmouseup`, `onmouseover`, `onmousemove`, `onmouseout`, `onkeypress`, `onkeydown`, `onkeyup` (intrinsic events)

The `LABEL` element may be used to attach information to other control elements (excluding other `LABEL` elements). Labels may be rendered by user agents in a number of ways (e.g., visually, read by speech synthesizers, etc.)

When a `LABEL` element receives focus, it passes the focus on to its associated control. See the section below on access keys for examples.

To associate a label with another control explicitly, set the `for` attribute of the `LABEL`.

This example creates a table that is used to align two `INPUT` controls and their associated labels. Each label is associated explicitly with one of the `INPUT` elements.

```
<FORM action="..." method="post">
<TABLE>
  <TR>
    <TD><LABEL for="fname">First Name</LABEL>
    <TD><INPUT type="text" name="firstname" id="fname">
  <TR>
    <TD><LABEL for="lname">Last Name</LABEL>
    <TD><INPUT type="text" name="lastname" id="lname">
</TABLE>
<FORM>
```

This example extends a previous example form to include `LABEL` elements. Note that the `LABEL` elements are associated to the `INPUT` elements through the `id` attribute.

```
 <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">
 </FORM>
```

More than one `LABEL` may be associated with the same control by creating multiple references via the `for` attribute.

To associate a label with another control implicitly, make the control the contents of the `LABEL`. In this case, the `LABEL` may only contain one other control element. The label itself may be positioned before or after the associated control.

In this example, we implicitly associate two labels and two `INPUT` elements. Notice that the implicit association prevents us from being able to layout the label and its associated control in a table (see the previous example).

```
<FORM action="..." method="post">
<LABEL>
   First Name
   <INPUT type="text" name="firstname">
</LABEL>
<LABEL>
   <INPUT type="text" name="lastname">
   Last Name
</LABEL>
</FORM>
```

The `FIELDSET` and `LEGEND` elements

```
<!--
  #PCDATA is to solve the mixed content problem,
  per specification only whitespace is allowed there!
 -->
<!ELEMENT FIELDSET - - (#PCDATA,LEGEND,%block)>
<!ATTLIST FIELDSET
  %attrs;                          -- %coreattrs, %i18n, %events --
  >

<!ELEMENT LEGEND - - (%inline;)+>
<!ENTITY % LAlign "(top|bottom|left|right)">

<!ATTLIST LEGEND            -- fieldset legend --
  %attrs;                          -- %coreattrs, %i18n, %events --
  align       %LAlign;   #IMPLIED  -- relative to fieldset --
  accesskey   CDATA      #IMPLIED  -- accessibility key character --
  >
```

Start tag: ***required****, End tag:* ***required***

LEGEND Attribute definitions`align = top|bottom|left|right`
This attribute specifies the position of the legend with respect to the fieldset. Possible values:

- `top:` The legend is above the fieldset. This is the default value.
- `bottom:` The legend is below the fieldset.
- `left:` The legend is to the left of the fieldset.
- `right:` The legend is to the right of the fieldset.

Attributes defined elsewhere

- `id`, `class` (document-wide identifiers)
- `lang` (language information), `dir` (text direction)
- `title` (element titles)
- `style` (inline style information)
- `accesskey` (access keys)
- `align` (alignment)
- `onclick`, `ondblclick`, `onmousedown`, `onmouseup`, `onmouseover`, `onmousemove`, `onmouseout`, `onkeypress`, `onkeydown`, `onkeyup` (intrinsic events)

The `FIELDSET` element allows form designers to group thematically related controls together. 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 to people with disabilities.

The `LEGEND` element allows designers to assign a caption to a `FIELDSET`. The legend improves accessibility when the `FIELDSET` is rendered non-visually. When rendered visually, setting the `align` attribute on the `LEGEND` element aligns it with respect to the `FIELDSET`.

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">
<FIELDSET>
<LEGEND align="top">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 align="top">Medical History</LEGEND>
<INPUT name="history_illness"
       type="checkbox"
       value="Smallpox" tabindex="20"> Smallpox</INPUT>
<INPUT name="history_illness"
       type="checkbox"
       value="Mumps" tabindex="21"> Mumps</INPUT>
<INPUT name="history_illness"
       type="checkbox"
       value="Dizziness" tabindex="22"> Dizziness</INPUT>
<INPUT name="history_illness"
       type="checkbox"
       value="Sneezing" tabindex="23"> Sneezing</INPUT>
```

```
...more medical history...
</FIELDSET>
<FIELDSET>
<LEGEND align="top">Current Medication</LEGEND>
Are you currently taking any medication?
<INPUT name="medication_now"
       type="radio"
       value="Yes" tabindex="35">Yes</INPUT>
<INPUT name="medication_now"
       type="radio"
       value="No" tabindex="35">No</INPUT>

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 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.

Giving focus to an element

Active elements in HTML documents must receive *focus* from the user in order to perform their 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 (tabbing navigation). Once selected, an element may be activated by some other key sequence.
- Select an element by a series of keyboard actions known as an *access key* (sometimes called "keyboard shortcut" or "keyboard accelerator").

Tabbing navigation

Attribute definitions

`tabindex` = *integer*
: This attribute specifies the position of the current element in the tabbing order for the current document. This value may be a positive or negative integer.

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 postive 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` should be navigated in the order they appear in the document.
2. Those elements that do not define the `tabindex` attribute or do not support it are navigated next. These elements are navigated in the order they appear in the document.
3. Those elements that support the `tabindex` attribute and assign a negative value to it do not participate in the tabbing order.
4. Elements that are disabled do not participate in the tabbing order.

The following elements support the `tabindex` attribute: `A`, `AREA`, `OBJECT`, `INPUT`, `SELECT`, `TEXTAREA`, and `BUTTON`.

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 document), and finally the link created by the `A` element.

```
<HTML>
<BODY>
...some text...
Click to go to the
<A tabindex="10" href="http://www.w3.org/">W3C Website.</A>
...some more...
<BUTTON type="button" name="get-database"
           tabindex="1" onclick="get-database">
Click me to receive the current database.
</BUTTON>
...some more...
<FORM action="..." method="post">
<INPUT tabindex="1" type="text" name="field1">
<INPUT tabindex="2" type="text" name="field2">
<INPUT tabindex="3" type="submit" name="submit">
</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).

Access keys

Attribute definitions

`accesskey` = *cdata*
: This attribute assigns an access key to an element. An access key is a single character from the user agent's current character encoding. User agents should treat the value of this attribute as case-insensitive.

Pressing an access key assigned to an element gives focus to the element. The action that is executed when an element receives focus depends on the element. Links defined by `A` are generally followed by the user agent, activated radio buttons change values, text fields with focus allow user input, etc.

The following elements support the `accesskey` attribute: `LABEL`, `A`, `CAPTION`, and `LEGEND`.

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">
<LABEL for="user" accesskey="U">
User Name
</LABEL>
<INPUT type="text" name="user">
</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.

```
<A accesskey="C"
      href="http://somplace.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).

Disabled and read-only elements

In contexts where user input is either undesirable or irrelevant, it is important to be able to disable an element 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 elements.

Disabled elements

Attribute definitions

`disabled`
: 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 elements do not receive focus.
- Disabled elements are skipped in tabbing navigation.
- Values of disabled controls are not submitted with a form.

The following elements support the `disabled` attribute: `INPUT`, `TEXTAREA`, `SELECT`, `OPTION`, `OBJECT`, `LABEL`, and `BUTTON`.

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 disabled `INPUT` element 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.

Read-only elements

Attribute definitions

`readonly`
: When set for a form control, this boolean attribute prohibits changes to control.

The `readonly` attribute specifies whether the element may be modified by the user.

When set, the `readonly` attribute has the following effects on an element:

- Read-only elements receive focus but cannot be modified by the user.
- Read-only elements are included in tabbing navigation.
- Values of read-only controls are submitted with a form.

The following elements support the `readonly` attribute: `INPUT`, `TEXT`, `PASSWORD`, 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.

Form submission

Which element values are submitted

Not all elements have their values submitted with a form. Conforming user agents should **not** submit:

- Disabled form controls.
- Form controls without values for the `name` attribute.
- `OBJECT` elements without the `name` attribute.
- `OBJECT` elements with the `declare` attribute.

Scripts

Contents

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 document 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 attributes.

Note: This specification includes more detailed information about scripting in sections on script macros.

Designing documents for user agents that support scripting

The following sections discuss issues that concern user agents that support scripting.

The `SCRIPT` element

```
<!ELEMENT SCRIPT - - CDATA      -- script statements -->
<!ATTLIST SCRIPT
  type        CDATA      #IMPLIED  -- Internet content type for
                                      script language --
  language    CDATA      #IMPLIED  -- predefined script language name --
  src         %URL       #IMPLIED  -- URL for an external script --
  >
```

Start tag: ***required****, End tag:* ***required***

Attribute definitions

`type` = *cdata*
: This attribute specifies the scripting language of the contents of this element. The value must be an Internet Media Type. There is no default value for this attribute.

`language` = *cdata*
: **Deprecated.**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 in favor of `type`.

`src` = *url*
: This attribute specifies the location of an external script.

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 URL value, user agents must ignore the element's contents and retrieve the script via the URL.

Scripts are evaluated by *script engines* that must be known to a user agent.

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.

Documents that contain neither a default scripting language declaration nor a local one for a `SCRIPT` element are incorrect. User agents may still try to interpret the script but are not required to.

The default scripting language

To specify the default scripting language for all scripts in a document include the following `META` declaration in the `HEAD` of a document:

```
<META http-equiv="Content-Script-Type" content="type">
```

where "type" is an Internet Media Type (see [MIMETYPES]) naming the scripting language. Examples of values include "text/tcl", "text/javascript", "text/vbscript". See [MIMETYPES] for a complete list of valid scripting language types.

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 an Internet Media Type naming the scripting language.

When several HTTP headers and META elements occur, the last one defines the default scripting language. For our purposes, HTTP headers are considered to occur earlier than the document HEAD.

Local declaration of a scripting language

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. When a default scripting language has been specified, the type attribute overrides it.

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".

```
<HTML>
<HEAD>
<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 a id if both are set. Otherwise, one or the other may be used.

Syntax of script content

The content of the SCRIPT element is a script, and as such, must not be evaluated by the user agent as HTML markup. The user agent must pass it on as data to a script engine.

HTML parsers must be able to recognize script data as beginning immediately after the start tag and ending as soon as the ETAGO ("</") delimiters are followed by a name character ([a-zA-Z]). The script data does not necessarily end with the </SCRIPT> end tag, but is terminated by any "</" followed by a name character.

Consequently, any HTML markup that is meant to be sent to a script engine (which may do whatever it wants with the markup) must be "escaped" so as not to confuse the HTML parser. Designers of each scripting language should recommend language-specific support for resolving this issue.

ILLEGAL EXAMPLE:
The following code is invalid due the to presence of the "</EM>" characters found inside of the SCRIPT element:

```
<SCRIPT type="text/javascript">
  document.write ("<EM>This won't work</EM>")
</SCRIPT>
```

A conforming parser must treat the "</EM>" data as the end of script data, which is clearly not what the author intended.

In JavaScript, this code can be expressed legally by ensuring that the apparent ETAGO delimiter does not appear immediately 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>"
```

Intrinsic events

Attribute definitions

`onload` = *script*
: 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*
: 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*
: The `onclick` event occurs when the pointing device button is clicked over an element. This attribute may be used with most elements.

`ondblclick` = *script*
: 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*
: The `onmousedown` event occurs when the pointing device button is pressed over an element. This attribute may be used with most elements.

`onmouseup` = *script*
: The `onmouseup` event occurs when the pointing device button is released over an element. This attribute may be used with most elements.

`onmouseover` = *script*
: The `onmouseover` event occurs when the pointing device is moved over an element. This attribute may be used with most elements.

`onmousemove` = *script*
: The `onmousemove` event occurs when the pointing device is moved over an element. This attribute may be used with most elements.

`onmouseout` = *script*
: The `onmouseout` event occurs when the pointing device is moved away from an element. This attribute may be used with most elements.

`onfocus` = *script*
: 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: `LABEL`, `INPUT`, `SELECT`, `TEXTAREA`, and `BUTTON`.

`onblur` = *script*
: 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*
: The `onkeypress` event occurs when a key is pressed and released over an element. This attribute may be used with most elements.

`onkeydown` = *script*
: The `onkeydown` event occurs when a key is pressed down over an element. This attribute may be used with most elements.

`onkeyup` = *script*
: The `onkeyup` event occurs when a key is released over an element. This attribute may be used with most elements.

`onsubmit` = *script*
: The `onsubmit` event occurs when a form is submitted. It only applies to the `FORM` element.

`onreset` = *script*
: The `onreset` event occurs when a form is reset. It only applies to the `FORM` element.

`onselect` = *script*
: 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*

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.

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, designers 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)">
```

Here is another JavaScript example:

```
<INPUT NAME="num"
    onChange="if (!checkNum(this.value, 1, 10))
        {this.focus();this.select();} else {thanks()}"
    VALUE="0">
```

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/URL")
      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/URL"]
     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.

Parsing of intrinsic event scripts

The script attributes for intrinsic events are defined as CDATA. The SGML processing of CDATA attribute values requires that (1) entity replacement occur within the attribute value; and (2) that the attribute value be delimited by matching pairs of double quotes (") or single quotes (’).

Given these lexical restrictions, the delimiters (’), ("), "&", and "&#" may not occur freely in the value of a script attribute. To resolve this issue, we recommend that script event handler attributes always use (") delimiters and that occurrences of (") and "&" inside an event handler attribute be written as follows:

```
’"’  should be written as "&quot;" or as "&#34;"
’&’  should be written as "&amp;"  or as "&#38;"
```

Thus, for example, one could write:

```
<INPUT name="num" value="0"
onChange="if (compare(this.value, &quot;help&quot;)) {gethelp()}">
```

SGML permits (’) to be included in attribute strings quoted by ("), and vice versa. The following is therefore correct:

```
"this is ’fine’" and ’so is "this"’
```

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 evaluted. 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>
```

Designing documents for user agents that don't support scripting

The following sections discuss issues about lack of support for scripting that authors should also consider when designing good HTML documents.

The **NOSCRIPT** element

```
<!ELEMENT NOSCRIPT - - (%block)>
```

*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 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>To access the data, click <A href="http://someplace.com/data">here.</A>
</NOSCRIPT>
```

Commenting out scripts

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 "<!--" 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 "-->" from the JavaScript parser.

```
<SCRIPT type="text/javascript">
<!--  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 "-->" from VBScript, for instance:

```
   <SCRIPT type="text/vbscript">
     <!--
       Sub foo()
        ...
       End Sub
     ' -->
    </SCRIPT>
```

Commenting scripts in TCL
In Tcl, the "#" character comments out the rest of the line:

```
<SCRIPT type="text/tcl">
<!--  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 ">" 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 ">".

SGML reference information for HTML

Contents

The following sections contain the formal SGML definition of HTML 4.0, including the SGML declaration and the Document Type Definition (DTD), as well as a sample SGML catalog.

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.

The following sample SGML catalog can be used with an SGML parser, such as nsgmls, to verify that HTML documents conform to the HTML 4.0 DTD. It assumes that the DTD has been saved as the file "HTML4.dtd" and that the entities are in the files "HTMLlat1.ent", "HTMLsymbol.ent" and "HTMLspecial.ent". 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 4.0 specification. This is because an SGML parser relies solely on the given SGML DTD which does not express all aspects of a valid HTML 4.0 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", "12.5", or "25%"). 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, which actually allows any value. Only a specialized program could capture the complete specification of HTML 4.0.

Nevertheless, this type of validation is still highly recommended since it permits the detection of a large set of errors that make documents invalid.

Sample SGML catalog

```
PUBLIC "-//W3C//DTD HTML 4.0 Draft//EN" HTML4.dtd
PUBLIC "-//W3C//DTD HTML 4.0 Final//EN" HTML4.dtd
PUBLIC "-//W3C//DTD HTML 4.0//EN" HTML4.dtd
PUBLIC "-//W3C//ENTITIES Latin1//EN//HTML" ISOlat1.ent
PUBLIC "-//W3C//ENTITIES Special//EN//HTML" HTMLmisc.ent
PUBLIC "-//W3C//ENTITIES Symbols//EN//HTML" HTMLsym.ent
```

SGML Declaration

Contents

The Document Character Set

The HTML 4.0 document character set, in the SGML sense, is the Universal Character Set (UCS) of [ISO10646]. Currently, this is code-by-code identical with the [UNICODE] standard.

Data transfer

When HTML text is transmitted directly in UCS-2 (charset="UNICODE-1-1"), one must address the question of byte order: does the high-order byte of each two-byte character come first or second? This specification recommends that the UCS-2 be transmitted in big-endian byte order (high order byte first), which corresponds both to the established network byte order for two-byte quantities and to the Unicode ([UNICODE]) recommendation for serialized text data. Furthermore, to maximize chances of proper interpretation, it is recommended that documents transmitted as UCS-2 always begin with a ZERO-WIDTH NON-BREAKING SPACE character (hexadecimal FEFF) which, when byte-reversed becomes number FFFE, a character guaranteed to be never assigned. Thus, a user-agent receiving an FFFE as the first octets of a text would know that bytes have to be reversed for the remainder of the text.

The UTF-1 transformation format of [ISO10646] (registered by IANA as ISO-10646-UTF-1), should not be used.

The SGML Declaration

```
<!SGML  "ISO 8879:1986"
--
     SGML Declaration for HyperText Markup Language version 4.0

     With support for Unicode UCS-4 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 2147483486 160
--
    In ISO 10646, the positions with hexadecimal
    values 0000D800 - 0000DFFF, used in the UTF-16
    encoding of UCS-4, are reserved, as well as the last
    two code values in each plane of UCS-4, i.e. all
    values of the hexadecimal form xxxxFFFE or xxxxFFFF.
    These code values or the corresponding numeric
    character references must not be included when
    generating a new HTML document, and they should be
    ignored if encountered when processing a HTML
    document.
--

CAPACITY        SGMLREF
                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 SEPCHAR    9

         NAMING   LCNMSTRT ""
                  UCNMSTRT ""
                  LCNMCHAR ".-"  -- ?include "~/_" for URLs? --
                  UCNMCHAR ".-"
                  NAMECASE GENERAL YES
                           ENTITY  NO
         DELIM    GENERAL  SGMLREF
                  SHORTREF SGMLREF
         NAMES    SGMLREF
         QUANTITY SGMLREF
                  ATTSPLEN 65536   -- These are the largest values --
                  LITLEN   65536   -- permitted in the declaration --
                  NAMELEN  65536   -- Avoid fixed limits in actual --
                  PILEN    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
>
```

Document Type Definition

```
<!--
        This is an EXPERIMENTAL version of HTML 4.0 that
        extends HTML 3.2 to add support for proposals under review by
        the W3C HTML Working Group, including style sheets, scripting,
        the object tag, internationalization and some extensions to
        forms for improved accessibility by people with disabilities.
        The frame tags have been added in acknowledgement of their
        widespread deployment.

            Draft: $Date: 1997/07/08 10:47:55 $

            Authors:
                Dave Raggett <dsr@w3.org>
                Arnaud Le Hors <lehors@w3.org>

        This is work in progress, subject to change at any time.
        It does not imply endorsement by, or the consensus of,
        either W3C or members of the HTML working group. Further
        information about HTML 4.0 is available at:

            http://www.w3.org/TR/WD-html40-970708/.
-->
<!ENTITY % HTML.Version "http://www.w3.org/TR/WD-html40-970708/HTML4.dtd"
  -- Typical usage:

    <!DOCTYPE HTML SYSTEM "http://www.w3.org/TR/WD-html40-970708/HTML4.dtd">
    <html>
    ...
    </html>
  --
>

<!--================== Imported Names ====================================-->

<!ENTITY % ContentType "CDATA"
    -- an Internet media type, as per [RFC2045]
    -->

<!ENTITY % URL "CDATA"
    -- a Uniform Resource Locator,
       see [RFC1808] and [RFC1738]
    -->

<!ENTITY % Script "CDATA" -- scriptlet -->

<!-- Parameter Entities -->

<!ENTITY % head.misc "SCRIPT|STYLE|META|LINK" -- repeatable head elements -->

<!ENTITY % heading "H1|H2|H3|H4|H5|H6">

<!ENTITY % list "UL | OL |  DIR | MENU">

<!ENTITY % preformatted "PRE">
```

```
<!--================ Character mnemonic entities =========================-->

<!ENTITY % HTMLlat1 PUBLIC
   "-//W3C//ENTITIES Latin1//EN//HTML">
%HTMLlat1;

<!ENTITY % HTMLsymbol PUBLIC
   "-//W3C//ENTITIES Symbols//EN//HTML">
%HTMLsymbol;

<!ENTITY % HTMLspecial PUBLIC
   "-//W3C//ENTITIES Special//EN//HTML">
%HTMLspecial;

<!--=================== Generic Attributes ===============================-->

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

<!ENTITY % i18n
 "lang        NAME        #IMPLIED  -- [RFC1766] language value --
  dir         (ltr|rtl)   #IMPLIED  -- direction for weak/neutral text --"
  >

<!ENTITY % events
 "onclick     %Script     #IMPLIED  -- the pointing device button was clicked --
  ondblclick  %Script     #IMPLIED  -- the pointing device button was double clicked --
  onmousedown %Script     #IMPLIED  -- the pointing device button was pressed down --
  onmouseup   %Script     #IMPLIED  -- the pointing device button was released --
  onmouseover %Script     #IMPLIED  -- the pointing device was moved over --
  onmousemove %Script     #IMPLIED  -- the pointing device was moved --
  onmouseout  %Script     #IMPLIED  -- the pointing device 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 --"
  >


<!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 % font
 "TT | I | B | U | S | STRIKE | BIG | SMALL">

<!ENTITY % phrase "EM | STRONG | DFN | CODE |
                   SAMP | KBD | VAR | CITE | 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 | %font | %phrase | %special | %formctrl">

<!ELEMENT (%font|%phrase) - - (%inline)*>
<!ATTLIST (%font|%phrase)
  %attrs;                          -- %coreattrs, %i18n, %events --
  >

<!-- subscripts and superscripts -->
<!ELEMENT (SUB|SUP) - - (%inline)*>
<!ATTLIST (SUB|SUP)
  %attrs;                          -- %coreattrs, %i18n, %events --
  >

<!ELEMENT SPAN - - (%inline)*     -- generic language/style container -->
<!ATTLIST SPAN
  %attrs;                          -- %coreattrs, %i18n, %events --
  >

<!-- INS/DEL are handled by inclusion on BODY -->
<!ELEMENT (INS|DEL) - - (%inline)* -- inserted/deleted text -->
<!ATTLIST (INS|DEL)
  %attrs                           -- %coreattrs, %i18n, %events --
  cite        %URL      #IMPLIED   -- info on reason for change --
  datetime    CDATA     #IMPLIED   -- when changed: ISO date format --
  >

<!ELEMENT BDO - - (%inline)*      -- I18N BiDi over-ride -->
<!ATTLIST BDO
  lang        NAME       #IMPLIED  -- [RFC1766] language value --
  dir         (ltr|rtl)  #REQUIRED -- directionality --
  >

<!ELEMENT BASEFONT - O EMPTY>
<!ATTLIST BASEFONT
  size        CDATA      #REQUIRED -- base font size for FONT elements --
  color       CDATA      #IMPLIED  -- #RRGGBB in hex, e.g. red: "#FF0000" --
  face        CDATA      #IMPLIED  -- comma separated list of font names --
  >

<!ELEMENT FONT - - (%inline)*   -- local change to font -->
<!ATTLIST FONT
  size        CDATA      #IMPLIED  -- [+]nn e.g. size="+1", size=4 --
  color       CDATA      #IMPLIED  -- #RRGGBB in hex, e.g. red: "#FF0000" --
  face        CDATA      #IMPLIED  -- comma separated list of font names --
  >

<!ELEMENT BR - O EMPTY          -- forced line break -->
<!ATTLIST BR
  %coreattrs;                      -- id, class, style, title --
  clear (left|all|right|none) none -- control of text flow --
```

```
  >

<!--================== 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 % blocklevel
     "P | %heading | %list | %preformatted | DL | DIV | CENTER |
      NOSCRIPT | NOFRAMES | BLOCKQUOTE | FORM | ISINDEX | HR |
      TABLE | FIELDSET | ADDRESS">

<!--=================== Document Body ====================================-->

<!ENTITY % block "(%blocklevel | %inline)*">

<!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
  text  %Color   #IMPLIED
  link  %Color   #IMPLIED
  vlink %Color   #IMPLIED
  alink %Color   #IMPLIED
  ">

<!ELEMENT BODY O O  (%block) -(BODY) +(INS|DEL)>
<!ATTLIST BODY
  %attrs;                         -- %coreattrs, %i18n, %events --
  background  %URL       #IMPLIED  -- texture tile for document background --
  %bodycolors;                    -- bgcolor, text, link, vlink, alink --
  onload      %Script    #IMPLIED  -- the document has been loaded --
  onunload    %Script    #IMPLIED  -- the document has been removed --
  >

<!ELEMENT ADDRESS - - ((%inline;) | P)*>
<!ATTLIST ADDRESS
  %attrs;                         -- %coreattrs, %i18n, %events --
  >

<!ELEMENT DIV - - %block>
```

```
<!ATTLIST DIV
  %attrs;                          -- %coreattrs, %i18n, %events --
  %align;                          -- align, text alignment --
  >

<!-- CENTER is a shorthand for DIV with ALIGN=CENTER -->
<!ELEMENT CENTER - - %block>
<!ATTLIST CENTER
  %attrs;                          -- %coreattrs, %i18n, %events --
  >

<!--================== The Anchor Element ================================-->

<!ENTITY % Shape "(rect|circle|poly|default)">
<!ENTITY % Coords "CDATA" -- comma separated list of numbers -->

<!ELEMENT A - - (%inline)* -(A)>
<!ATTLIST A
  %attrs;                          -- %coreattrs, %i18n, %events --
  charset     CDATA      #IMPLIED  -- char encoding of linked resource --
  name        CDATA      #IMPLIED  -- named link end --
  href        %URL       #IMPLIED  -- URL for linked resource --
  target      CDATA      #IMPLIED  -- where to render resource --
  rel         CDATA      #IMPLIED  -- forward link types --
  rev         CDATA      #IMPLIED  -- reverse link types --
  accesskey   CDATA      #IMPLIED  -- accessibility key character --
  shape       %Shape     rect      -- for use with OBJECT SHAPES --
  coords      %Coords    #IMPLIED  -- for use with OBJECT SHAPES --
  tabindex    NUMBER     #IMPLIED  -- position in tabbing order --
  >

<!--================== 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 - - (AREA)*>
<!ATTLIST MAP
  %coreattrs;                      -- id, class, style, title --
  name        CDATA     #IMPLIED
  >

<!ELEMENT AREA - O EMPTY>
<!ATTLIST AREA
  shape       %Shape    rect      -- controls interpretation of coords --
  coords      %Coords   #IMPLIED  -- comma separated list of values --
  href        %URL      #IMPLIED  -- this region acts as hypertext link --
  target      CDATA     #IMPLIED  -- where to render linked resource --
  nohref      (nohref)  #IMPLIED  -- this region has no action --
  alt         CDATA     #REQUIRED -- description for text only browsers --
  tabindex    NUMBER    #IMPLIED  -- position in tabbing order --
  >

<!--================== 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=print)
-->

<!ELEMENT LINK - O EMPTY>
<!ATTLIST LINK
  %attrs;                        -- %coreattrs, %i18n, %events --
  href        %URL       #IMPLIED  -- URL for linked resource --
  rel         CDATA      #IMPLIED  -- forward link types --
  rev         CDATA      #IMPLIED  -- reverse link types --
  type    %ContentType  #IMPLIED  -- advisory Internet content type --
  media       CDATA      #IMPLIED  -- for rendering on these media --
  target      CDATA      #IMPLIED  -- where to render linked resource --
  >

<!--=================== Images ===========================================-->

<!ENTITY % Length "CDATA" -- nn for pixels or nn% for percentage length -->
<!ENTITY % Pixels "CDATA" -- integer representing length in pixels -->

<!ENTITY % IAlign "(top|middle|bottom|left|right)"  -- center? -->

<!-- To avoid problems with text-only UAs you need to provide
     a description with ALT, and avoid server-side image maps -->
<!ELEMENT IMG - O EMPTY      --  Embedded image -->
<!ATTLIST IMG
  %attrs;                        -- %coreattrs, %i18n, %events --
  src         %URL       #REQUIRED -- URL of image to embed --
  alt         CDATA      #IMPLIED  -- description for text only browsers --
  align       %IAlign    #IMPLIED  -- vertical or horizontal alignment --
  height      %Pixels    #IMPLIED  -- suggested height in pixels --
  width       %Pixels    #IMPLIED  -- suggested width in pixels --
  border      %Pixels    #IMPLIED  -- suggested link border width --
  hspace      %Pixels    #IMPLIED  -- suggested horizontal gutter --
  vspace      %Pixels    #IMPLIED  -- suggested vertical gutter --
  usemap      %URL       #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 tag ======================================-->
<!-- PARAM elements should precede other content. SGML mixed content model
  technicality precludes specifying this formally ...
-->

<!ENTITY % OAlign "(texttop|middle|textmiddle|baseline|
                             textbottom|left|center|right)">
```

```
<!ELEMENT OBJECT - - (PARAM | %block)*>
<!ATTLIST OBJECT
  %attrs                         -- %coreattrs, %i18n, %events --
  declare     (declare)  #IMPLIED  -- declare but don't instantiate flag --
  classid     %URL       #IMPLIED  -- identifies an implementation --
  codebase    %URL       #IMPLIED  -- some systems need an additional URL --
  data        %URL       #IMPLIED  -- reference to object's data --
  type     %ContentType #IMPLIED  -- Internet content type for data --
  codetype %ContentType #IMPLIED  -- Internet content type for code --
  standby     CDATA      #IMPLIED  -- message to show while loading --
  align       %OAlign    #IMPLIED  -- positioning inside document --
  height      %Length    #IMPLIED  -- suggested height --
  width       %Length    #IMPLIED  -- suggested width --
  border      %Length    #IMPLIED  -- suggested link border width --
  hspace      %Length    #IMPLIED  -- suggested horizontal gutter --
  vspace      %Length    #IMPLIED  -- suggested vertical gutter --
  usemap      %URL       #IMPLIED  -- reference to image map --
  shapes      (shapes)   #IMPLIED  -- object has shaped hypertext links --
  name        %URL       #IMPLIED  -- submit as part of form --
  tabindex    NUMBER     #IMPLIED  -- position in tabbing order --
  >

<!ELEMENT PARAM - O EMPTY       -- named property value -->
<!ATTLIST PARAM
  name        CDATA      #REQUIRED -- property name --
  value       CDATA      #IMPLIED  -- property value --
  valuetype (DATA|REF|OBJECT) DATA -- How to interpret value --
  type        CDATA      #IMPLIED  -- Internet media type --
  >

<!--=================== Java APPLET tag ==================================-->
<!--
  One of code or object attributes must be present.
  Place PARAM elements before other content.
-->
<!ELEMENT APPLET - - (PARAM | %inline)*>
<!ATTLIST APPLET
  codebase    %URL       #IMPLIED   -- optional base URL for applet --
  archive     CDATA      #IMPLIED   -- comma separated archive list --
  code        CDATA      #IMPLIED   -- applet class file --
  object      CDATA      #IMPLIED   -- serialized applet file --
  alt         CDATA      #IMPLIED   -- description for text only browsers --
  name        CDATA      #IMPLIED   -- allows applets to find each other --
  width       %Pixels    #REQUIRED  -- suggested width in pixels --
  height      %Pixels    #REQUIRED  -- suggested height in pixels --
  align       %IAlign    #IMPLIED   -- vertical or horizontal alignment --
  hspace      %Pixels    #IMPLIED   -- suggested horizontal gutter --
  vspace      %Pixels    #IMPLIED   -- suggested vertical gutter --
  >

<!--=================== Horizontal Rule ==================================-->

<!ELEMENT HR - O EMPTY>
<!ATTLIST HR
  %coreattrs;                    -- id, class, style, title --
  %events;
```

```
  align (left|right|center) #IMPLIED
  noshade     (noshade)  #IMPLIED
  size        %Pixels    #IMPLIED
  width       %Length    #IMPLIED
  >

<!--=================== Paragraphs =======================================-->

<!ELEMENT P - O (%inline)*>
<!ATTLIST P
  %attrs;                          -- %coreattrs, %i18n, %events --
  %align;                          -- align, text alignment --
  >

<!--=================== Headings =========================================-->

<!--
  There are six levels of headings from H1 (the most important)
  to H6 (the least important).
-->

<!ELEMENT (%heading)  - -  (%inline;)*>
<!ATTLIST (%heading)
  %attrs;                          -- %coreattrs, %i18n, %events --
  %align;                          -- align, text alignment --
  >

<!--=================== Preformatted Text ================================-->

<!-- excludes markup for images and changes in font size -->
<!ENTITY % pre.exclusion "IMG|BIG|SMALL|SUB|SUP|FONT">

<!ELEMENT PRE - - (%inline)* -(%pre.exclusion)>
<!ATTLIST PRE
  %attrs;                          -- %coreattrs, %i18n, %events --
  width       NUMBER     #IMPLIED
  >

<!--===================== Inline Quotes ==================================-->

<!ELEMENT Q - - (%inline)*>
<!ATTLIST Q
  %attrs;                          -- %coreattrs, %i18n, %events --
  cite        %URL       #IMPLIED  -- URL for source document or msg --
  >

<!--=================== Block-like Quotes ================================-->

<!ELEMENT BLOCKQUOTE - - %block>
<!ATTLIST BLOCKQUOTE
  %attrs;                          -- %coreattrs, %i18n, %events --
  cite        %URL       #IMPLIED  -- URL for source document or msg --
  >

<!--=================== Lists ============================================-->

<!-- definition lists - DT for term, DD for its definition -->
```

```
<!ELEMENT DL - -  (DT|DD)+>
<!ATTLIST DL
  %attrs;                          -- %coreattrs, %i18n, %events --
  compact     (compact)  #IMPLIED  -- reduced interitem spacing --
  >

<!ELEMENT DT - O  (%inline)*>
<!ELEMENT DD - O  %block>
<!ATTLIST (DT|DD)
  %attrs                          -- %coreattrs, %i18n, %events --
  >

<!-- Ordered lists (OL) Numbering style

    1   arablic 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|A|i|I] -->

<!ELEMENT OL - -  (LI)+>
<!ATTLIST OL -- ordered lists --
  %attrs;                          -- %coreattrs, %i18n, %events --
  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)+>
<!ATTLIST UL                    -- unordered lists --
  %attrs;                          -- %coreattrs, %i18n, %events --
  type        (%ULStyle) #IMPLIED  -- bullet style --
  compact     (compact)  #IMPLIED  -- reduced interitem spacing --
  >

<!ELEMENT (DIR|MENU) - -  (LI)+ -(%blocklevel)>
<!ATTLIST DIR
  %attrs;                          -- %coreattrs, %i18n, %events --
  compact     (compact)  #IMPLIED
  >
<!ATTLIST MENU
  %attrs;                          -- %coreattrs, %i18n, %events --
  compact     (compact)  #IMPLIED
  >
```

```
<!-- <DIR>              Directory list                  -->
<!-- <DIR COMPACT>      Compact list style              -->
<!-- <MENU>             Menu list                       -->
<!-- <MENU COMPACT>     Compact list style              -->

<!-- The type attribute can be used to change the bullet style
     in unordered lists and the numbering style in ordered lists -->

<!ENTITY % LIStyle "CDATA" -- constrained to: "(%ULStyle|%OLStyle)" -->

<!ELEMENT LI - O %block -- list item -->
<!ATTLIST LI
  %attrs;                        -- %coreattrs, %i18n, %events --
  type        %LIStyle   #IMPLIED  -- list item style --
  value       NUMBER     #IMPLIED  -- reset sequence number --
  >


<!--================ Forms ===============================================-->
<!ELEMENT FORM - - %block -(FORM)>
<!ATTLIST FORM
  %attrs;                        -- %coreattrs, %i18n, %events --
  action      %URL       #REQUIRED -- server-side form handler --
  method      (GET|POST) GET       -- HTTP method used to submit the form --
  enctype     %ContentType; "application/x-www-form-urlencoded"
  onsubmit    %Script    #IMPLIED  -- the form was submitted --
  onreset     %Script    #IMPLIED  -- the form was reset --
  target      CDATA      #IMPLIED  -- where to render result --
  accept-charset CDATA   #IMPLIED  -- list of supported charsets --
  >

<!-- Each label must not contain more than ONE field -->
<!ELEMENT LABEL - - (%inline)* -(LABEL) -- field label text -->
<!ATTLIST LABEL
  %attrs;                        -- %coreattrs, %i18n, %events --
  for         IDREF      #IMPLIED  -- matches field ID value --
  disabled    (disabled) #IMPLIED  -- control is unavailable in this context --
  accesskey   CDATA      #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)"
   >

<!-- HSPACE and VSPACE missing due to lack of widespread support -->
<!ELEMENT INPUT - O EMPTY>
<!ATTLIST INPUT
  %attrs;                        -- %coreattrs, %i18n, %events --
  type      %InputType   TEXT      -- what kind of widget is needed --
  name        CDATA      #IMPLIED  -- required for all but submit & reset --
  value       CDATA      #IMPLIED  -- required for radio and checkboxes --
  checked   (checked)    #IMPLIED  -- for radio buttons and check boxes --
  disabled  (disabled)   #IMPLIED  -- control is 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         %URL       #IMPLIED  -- for fields with images --
  alt         CDATA      #IMPLIED  -- description for text only browsers --
  usemap      %URL       #IMPLIED  -- use client-side image map --
  align       %IAlign    #IMPLIED  -- vertical or horizontal alignment --
  tabindex    NUMBER     #IMPLIED  -- position in tabbing order --
  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 --
  accept      CDATA      #IMPLIED  -- list of MIME types for file upload --
  >

<!ELEMENT SELECT - - (OPTION+)>
<!ATTLIST SELECT
  %attrs;                          -- %coreattrs, %i18n, %events --
  name        CDATA      #REQUIRED -- field name --
  size        NUMBER     #IMPLIED  -- rows visible --
  multiple  (multiple)   #IMPLIED  -- default is single selection --
  disabled  (disabled)   #IMPLIED  -- control is 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 --
  onselect    %Script    #IMPLIED  -- some text was selected --
  onchange    %Script    #IMPLIED  -- the element value was changed --
  >

<!ELEMENT OPTION - O (#PCDATA)*>
<!ATTLIST OPTION
  %attrs;                          -- %coreattrs, %i18n, %events --
  selected  (selected)   #IMPLIED
  disabled  (disabled)   #IMPLIED  -- control is unavailable in this context --
  value       CDATA      #IMPLIED  -- defaults to element content --
  >

<!-- Multi-line text input field. -->
<!ELEMENT TEXTAREA - - (#PCDATA)*>
<!ATTLIST TEXTAREA
  %attrs;                          -- %coreattrs, %i18n, %events --
  name        CDATA      #REQUIRED
  rows        NUMBER     #REQUIRED
  cols        NUMBER     #REQUIRED
  disabled  (disabled)   #IMPLIED  -- control is unavailable in this context --
  readonly  (readonly)   #IMPLIED
  tabindex    NUMBER     #IMPLIED  -- position in tabbing order --
  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 --
  >

<!--
  #PCDATA is to solve the mixed content problem,
  per specification only whitespace is allowed there!
 -->
```

```
<!ELEMENT FIELDSET - - (#PCDATA,LEGEND,%block)>
<!ATTLIST FIELDSET
  %attrs;                          -- %coreattrs, %i18n, %events --
  >

<!ELEMENT LEGEND - - (%inline;)+>
<!ENTITY % LAlign "(top|bottom|left|right)">

<!ATTLIST LEGEND            -- fieldset legend --
  %attrs;                          -- %coreattrs, %i18n, %events --
  align       %LAlign;   #IMPLIED  -- relative to fieldset --
  accesskey   CDATA      #IMPLIED  -- accessibility key character --
  >

<!ELEMENT BUTTON - -
     (%inline | %blocklevel)* -(A | %formctrl | FORM | ISINDEX | FIELDSET)>
<!ATTLIST BUTTON
  %attrs;                          -- %coreattrs, %i18n, %events --
  name        CDATA      #IMPLIED  -- for scripting/forms as submit button --
  value       CDATA      #IMPLIED  -- gets passed to server when submitted --
  type    (submit|reset) #IMPLIED  -- for use as form submit/reset button --
  disabled   (disabled)  #IMPLIED  -- control is 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 --
  >


<!--======================= 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
 CALS to avoid a name clash with the VALIGN 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       CDATA      #IMPLIED  -- alignment char, e.g. char=':' --
   charoff    CDATA      #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;)+>
<!ELEMENT THEAD - O (TR+)>
<!ELEMENT TFOOT - O (TR+)>
<!ELEMENT TBODY O O (TR+)>
<!ELEMENT COLGROUP - O (col*)>
<!ELEMENT COL - O EMPTY>
<!ELEMENT TR - O (TH|TD)+>
<!ELEMENT (TH|TD) - O %block>

<!ATTLIST TABLE                 -- table element --
  %attrs;                         -- %coreattrs, %i18n, %events --
  align       %TAlign;   #IMPLIED  -- table position relative to window --
  bgcolor     %Color     #IMPLIED  -- background color for cells --
  width       CDATA      #IMPLIED  -- table width relative to window --
  cols        NUMBER     #IMPLIED  -- used for immediate display mode --
  border      CDATA      #IMPLIED  -- controls frame width around table --
  frame       %TFrame;   #IMPLIED  -- which parts of table frame to include --
  rules       %TRules;   #IMPLIED  -- rulings between rows and cols --
  cellspacing CDATA      #IMPLIED  -- spacing between cells --
  cellpadding CDATA      #IMPLIED  -- spacing within cells --
  >

<!ENTITY % CAlign "(top|bottom|left|right)">

<!ATTLIST CAPTION               -- table caption --
  %attrs;                         -- %coreattrs, %i18n, %events --
  align       %CAlign;   #IMPLIED  -- relative to table --
  >

<!--
COLGROUP groups a set of COL elements. It allows you to group
several columns together.
-->
<!ATTLIST COLGROUP
  %attrs;                         -- %coreattrs, %i18n, %events --
  span        NUMBER     1         -- default number of columns in group --
  width       CDATA      #IMPLIED  -- default width for enclosed COLs --
  %cellhalign;                    -- horizontal alignment in cells --
  %cellvalign;                    -- vertical alignment in cells --
  >
```

```
<!--
 COL elements define the alignment properties for cells in a given
 column or spanned 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
-->
<!ATTLIST COL                    -- column groups and properties --
  %attrs;                           -- %coreattrs, %i18n, %events --
  span        NUMBER     1          -- number of columns spanned by group --
  width       CDATA      #IMPLIED   -- column width specification --
  %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
    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)   -- table section --
  %attrs;                           -- %coreattrs, %i18n, %events --
  %cellhalign;                      -- horizontal alignment in cells --
  %cellvalign;                      -- vertical alignment in cells --
  >

<!ATTLIST TR                     -- table row --
  %attrs;                           -- %coreattrs, %i18n, %events --
  %cellhalign;                      -- horizontal alignment in cells --
  %cellvalign;                      -- vertical alignment in cells --
  bgcolor     %Color     #IMPLIED   -- background color for row --
  >

<!ATTLIST (TH|TD)                -- header or data cell --
  %attrs;                           -- %coreattrs, %i18n, %events --
  axis        CDATA      #IMPLIED   -- defaults to cell content --
  axes        CDATA      #IMPLIED   -- list of axis names --
  nowrap      (nowrap)   #IMPLIED   -- suppress word wrap --
  bgcolor     %Color     #IMPLIED   -- cell background color --
  rowspan     NUMBER     1          -- number of rows spanned by cell --
  colspan     NUMBER     1          -- number of cols spanned by cell --
  %cellhalign;                      -- horizontal alignment in cells --
  %cellvalign;                      -- vertical alignment in cells --
  >

<!--================== Document Frames ===================================-->

<!ELEMENT FRAMESET - - ((FRAMESET|FRAME)+ & NOFRAMES?)>
<!ATTLIST FRAMESET
```

```
  -- absolute pixel values, percentages or relative scales. --
  rows        CDATA      #IMPLIED  -- if not given, default is 1 row --
  cols        CDATA      #IMPLIED  -- if not given, default is 1 column --
  onload      %Script    #IMPLIED  -- all the frames have been loaded  --
  onunload    %Script    #IMPLIED  -- all the frames have been removed --
  >

<!-- reserved frame names start with "_" otherwise starts with letter -->
<!ELEMENT FRAME - O EMPTY>
<!ATTLIST FRAME
  name        CDATA      #IMPLIED  -- name of frame for targetting --
  src         %URL       #IMPLIED  -- source of frame content --
  frameborder (1|0)      1         -- request frame borders? --
  marginwidth %Pixels    #IMPLIED  -- margin widths in pixels --
  marginheight %Pixels   #IMPLIED  -- margin height in pixels --
  noresize    (noresize) #IMPLIED  -- allow users to resize frames? --
  scrolling (yes|no|auto) auto     -- scrollbar or none --
  >

<!ELEMENT IFRAME - - %block>
<!ATTLIST IFRAME
  name        CDATA      #IMPLIED  -- name of frame for targetting --
  src         %URL       #IMPLIED  -- source of frame content --
  frameborder (1|0)      1         -- request frame borders? --
  marginwidth %Pixels    #IMPLIED  -- margin widths in pixels --
  marginheight %Pixels   #IMPLIED  -- margin height in pixels --
  scrolling (yes|no|auto) auto     -- scrollbar or none --
  align       %IAlign    #IMPLIED  -- vertical or horizontal alignment --
  height      %Length    #IMPLIED  -- suggested height --
  width       %Length    #IMPLIED  -- suggested width --
  >

<!--
 The following is quite complicated because of the mixed
 content model. However it's actually only meant to contain
 either BODY or %block.
 -->
<!ELEMENT NOFRAMES - -
 (#PCDATA,((BODY,#PCDATA)|
          (((%blocklevel)|%font|%phrase|%special|%formctrl),%block)))>

<!--================ Document Head =======================================-->
<!-- %head.misc defined earlier on as "SCRIPT | STYLE | META | LINK" -->
<!ENTITY % head.content "TITLE & ISINDEX? & BASE?">

<!ELEMENT HEAD O O  (%head.content) +(%head.misc)>
<!ATTLIST HEAD
  %i18n;                           -- lang, dir --
  profile     %URL       #IMPLIED  -- named dictionary of meta info --
  >

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

```
<!ELEMENT ISINDEX - O EMPTY>
<!ATTLIST ISINDEX
  %coreattrs;                      -- id, class, style, title --
  %i18n;                           -- lang, dir --
  prompt      CDATA      #IMPLIED  -- prompt message -->

<!ELEMENT BASE - O EMPTY>
<!ATTLIST BASE
  href        %URL       #REQUIRED
  target      CDATA      #IMPLIED  -- where to render linked resource --
  >

<!ELEMENT META - O EMPTY        -- Generic Metainformation -->
<!ATTLIST META
  %i18n;                           -- lang, dir, for use with content string --
  http-equiv  NAME       #IMPLIED  -- HTTP response header name  --
  name        NAME       #IMPLIED  -- metainformation name --
  content     CDATA      #REQUIRED -- associated information --
  scheme      CDATA      #IMPLIED  -- select form of content --
  >

<!ELEMENT STYLE - - CDATA      -- style info -->
<!ATTLIST STYLE
  %i18n;                           -- lang, dir, for use with title --
  type        CDATA      #REQUIRED -- Internet content type
                                      for style language --
  media       CDATA      #IMPLIED  -- designed for use with these media --
  title       CDATA      #IMPLIED  -- advisory title --
  >


<!ELEMENT SCRIPT - - CDATA      -- script statements -->
<!ATTLIST SCRIPT
  type        CDATA      #IMPLIED  -- Internet content type for
                                      script language --
  language    CDATA      #IMPLIED  -- predefined script language name --
  src         %URL       #IMPLIED  -- URL for an external script --
  >

<!ELEMENT NOSCRIPT - - (%block)>

<!--================ Document Structure ==================================-->
<!ENTITY % version "version CDATA #FIXED '%HTML.Version;'">

<!ENTITY % html.content "HEAD, (FRAMESET|BODY)">

<!ELEMENT HTML O O (%html.content)>
<!ATTLIST HTML
  %version;
  %i18n;                           -- lang, dir --
  >
```

Named character entities

Contents

A character entity is an SGML construct that references a character of the document character set. The document character set for HTML is the Universal Character Set (UCS) of [ISO10646]. This set is character-by-character equivalent to Unicode 2.0 ([UNICODE]).

This version of HTML supports several sets of named character entities:

- ISO 8859-1 (Latin-1) characters In accordance with section 14 of [RFC1866], 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 ([ISO8879]).
- symbols, mathematical symbols, and Greek letters. These characters may be represented by glyphs in the Adobe font "Symbol".
- arkup-significant and internationalization characters (e.g., for bidirectional text).

The following sections present the complete lists of named character entities. Although by convention, [ISO10646] names are written with upper case letters, we have converted them to lower case in this specification for reasons of readability.

Named entities for ISO 8859-1 characters

The named character entities in this section produce characters whose numeric equivalents should already be supported by conforming HTML 2.0 user agents. Thus, the named character entity ÷ 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].

Character 65533 (FFFD 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.

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 Full Latin 1//EN//HTML">
     %HTMLlat1;
-->

<!ENTITY nbsp   CDATA " " -- no-break space -->
<!ENTITY iexcl  CDATA "&#161;" -- inverted exclamation mark -->
<!ENTITY cent   CDATA "&#162;" -- cent sign -->
<!ENTITY pound  CDATA "&#163;" -- pound sterling sign -->
<!ENTITY curren CDATA "&#164;" -- general currency sign -->
<!ENTITY yen    CDATA "&#165;" -- yen sign -->
<!ENTITY brvbar CDATA "&#166;" -- broken (vertical) bar -->
<!ENTITY sect   CDATA "&#167;" -- section sign -->
<!ENTITY uml    CDATA "&#168;" -- umlaut (dieresis) -->
<!ENTITY copy   CDATA "&#169;" -- copyright sign -->
<!ENTITY ordf   CDATA "&#170;" -- ordinal indicator, feminine -->
<!ENTITY laquo  CDATA "&#171;" -- angle quotation mark, left -->
<!ENTITY not    CDATA "&#172;" -- not sign -->
<!ENTITY shy    CDATA "&#173;" -- soft hyphen -->
<!ENTITY reg    CDATA "&#174;" -- registered sign -->
<!ENTITY macr   CDATA "&#175;" -- macron -->
<!ENTITY deg    CDATA "&#176;" -- degree sign -->
<!ENTITY plusmn CDATA "&#177;" -- plus-or-minus sign -->
<!ENTITY sup2   CDATA "&#178;" -- superscript two -->
<!ENTITY sup3   CDATA "&#179;" -- superscript three -->
<!ENTITY acute  CDATA "&#180;" -- acute accent -->
<!ENTITY micro  CDATA "&#181;" -- micro sign -->
<!ENTITY para   CDATA "&#182;" -- pilcrow (paragraph sign) -->
<!ENTITY middot CDATA "&#183;" -- middle dot -->
<!ENTITY cedil  CDATA "&#184;" -- cedilla -->
<!ENTITY sup1   CDATA "&#185;" -- superscript one -->
<!ENTITY ordm   CDATA "&#186;" -- ordinal indicator, masculine -->
<!ENTITY raquo  CDATA "&#187;" -- angle quotation mark, right -->
<!ENTITY frac14 CDATA "&#188;" -- fraction one-quarter -->
<!ENTITY frac12 CDATA "&#189;" -- fraction one-half -->
<!ENTITY frac34 CDATA "&#190;" -- fraction three-quarters -->
<!ENTITY iquest CDATA "&#191;" -- inverted question mark -->
<!ENTITY Agrave CDATA "&#192;" -- capital A, grave accent -->
<!ENTITY Aacute CDATA "&#193;" -- capital A, acute accent -->
<!ENTITY Acirc  CDATA "&#194;" -- capital A, circumflex accent -->
<!ENTITY Atilde CDATA "&#195;" -- capital A, tilde -->
<!ENTITY Auml   CDATA "&#196;" -- capital A, dieresis or umlaut mark -->
<!ENTITY Aring  CDATA "&#197;" -- capital A, ring -->
<!ENTITY AElig  CDATA "&#198;" -- capital AE diphthong (ligature) -->
<!ENTITY Ccedil CDATA "&#199;" -- capital C, cedilla -->
<!ENTITY Egrave CDATA "&#200;" -- capital E, grave accent -->
<!ENTITY Eacute CDATA "&#201;" -- capital E, acute accent -->
<!ENTITY Ecirc  CDATA "&#202;" -- capital E, circumflex accent -->
```

```
<!ENTITY Euml   CDATA "&#203;" -- capital E, dieresis or umlaut mark -->
<!ENTITY Igrave CDATA "&#204;" -- capital I, grave accent -->
<!ENTITY Iacute CDATA "&#205;" -- capital I, acute accent -->
<!ENTITY Icirc  CDATA "&#206;" -- capital I, circumflex accent -->
<!ENTITY Iuml   CDATA "&#207;" -- capital I, dieresis or umlaut mark -->
<!ENTITY ETH    CDATA "&#208;" -- capital Eth, Icelandic -->
<!ENTITY Ntilde CDATA "&#209;" -- capital N, tilde -->
<!ENTITY Ograve CDATA "&#210;" -- capital O, grave accent -->
<!ENTITY Oacute CDATA "&#211;" -- capital O, acute accent -->
<!ENTITY Ocirc  CDATA "&#212;" -- capital O, circumflex accent -->
<!ENTITY Otilde CDATA "&#213;" -- capital O, tilde -->
<!ENTITY Ouml   CDATA "&#214;" -- capital O, dieresis or umlaut mark -->
<!ENTITY times  CDATA "&#215;" -- multiply sign -->
<!ENTITY Oslash CDATA "&#216;" -- capital O, slash -->
<!ENTITY Ugrave CDATA "&#217;" -- capital U, grave accent -->
<!ENTITY Uacute CDATA "&#218;" -- capital U, acute accent -->
<!ENTITY Ucirc  CDATA "&#219;" -- capital U, circumflex accent -->
<!ENTITY Uuml   CDATA "&#220;" -- capital U, dieresis or umlaut mark -->
<!ENTITY Yacute CDATA "&#221;" -- capital Y, acute accent -->
<!ENTITY THORN  CDATA "&#222;" -- capital THORN, Icelandic -->
<!ENTITY szlig  CDATA "&#223;" -- small sharp s, German (sz ligature) -->
<!ENTITY agrave CDATA "&#224;" -- small a, grave accent -->
<!ENTITY aacute CDATA "&#225;" -- small a, acute accent -->
<!ENTITY acirc  CDATA "&#226;" -- small a, circumflex accent -->
<!ENTITY atilde CDATA "&#227;" -- small a, tilde -->
<!ENTITY auml   CDATA "&#228;" -- small a, dieresis or umlaut mark -->
<!ENTITY aring  CDATA "&#229;" -- small a, ring -->
<!ENTITY aelig  CDATA "&#230;" -- small ae diphthong (ligature) -->
<!ENTITY ccedil CDATA "&#231;" -- small c, cedilla -->
<!ENTITY egrave CDATA "&#232;" -- small e, grave accent -->
<!ENTITY eacute CDATA "&#233;" -- small e, acute accent -->
<!ENTITY ecirc  CDATA "&#234;" -- small e, circumflex accent -->
<!ENTITY euml   CDATA "&#235;" -- small e, dieresis or umlaut mark -->
<!ENTITY igrave CDATA "&#236;" -- small i, grave accent -->
<!ENTITY iacute CDATA "&#237;" -- small i, acute accent -->
<!ENTITY icirc  CDATA "&#238;" -- small i, circumflex accent -->
<!ENTITY iuml   CDATA "&#239;" -- small i, dieresis or umlaut mark -->
<!ENTITY eth    CDATA "&#240;" -- small eth, Icelandic -->
<!ENTITY ntilde CDATA "&#241;" -- small n, tilde -->
<!ENTITY ograve CDATA "&#242;" -- small o, grave accent -->
<!ENTITY oacute CDATA "&#243;" -- small o, acute accent -->
<!ENTITY ocirc  CDATA "&#244;" -- small o, circumflex accent -->
<!ENTITY otilde CDATA "&#245;" -- small o, tilde -->
<!ENTITY ouml   CDATA "&#246;" -- small o, dieresis or umlaut mark -->
<!ENTITY divide CDATA "&#247;" -- divide sign -->
<!ENTITY oslash CDATA "&#248;" -- small o, slash -->
<!ENTITY ugrave CDATA "&#249;" -- small u, grave accent -->
<!ENTITY uacute CDATA "&#250;" -- small u, acute accent -->
<!ENTITY ucirc  CDATA "&#251;" -- small u, circumflex accent -->
<!ENTITY uuml   CDATA "&#252;" -- small u, dieresis or umlaut mark -->
<!ENTITY yacute CDATA "&#253;" -- small y, acute accent -->
<!ENTITY thorn  CDATA "&#254;" -- small thorn, Icelandic -->
<!ENTITY yuml   CDATA "&#255;" -- small y, dieresis or umlaut mark -->
```

Named entities for symbols, mathematical symbols, and Greek letters

The named character entities 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] or use other means. Display of glyphs for these characters may be obtained by being able to display the relevant [ISO10646] 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.*

The list of characters

```
<!-- Mathematical, Greek and Symbolic characters for HTML -->

<!-- Character entity set. Typical invocation:
     <!ENTITY % HTMLsymbol PUBLIC
       "-//W3C//ENTITIES Symbolic//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 Unicode 2.0 names.

-->

<!-- Latin Extended-B -->
<!ENTITY fnof     CDATA "&#402;"  -- latin small f with hook, =function, =florin, 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 ISOgrk3 -->
<!ENTITY Delta    CDATA "&#916;" -- greek capital letter delta,  u+0394 ISOgrk3 -->
<!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 ISOgrk3 -->
<!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 ISOgrk3 -->
<!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 ISOgrk3 -->
<!ENTITY Omicron  CDATA "&#927;" -- greek capital letter omicron,  u+039F -->
<!ENTITY Pi       CDATA "&#928;" -- greek capital letter pi,  u+03A0 ISOgrk3 -->
<!ENTITY Rho      CDATA "&#929;" -- greek capital letter rho,  u+03A1 -->
<!-- (there is no Sigmaf, and no u+03A2 character either) -->
<!ENTITY Sigma    CDATA "&#931;" -- greek capital letter sigma,  u+03A3 ISOgrk3 -->
<!ENTITY Tau      CDATA "&#932;" -- greek capital letter tau,  u+03A4 -->
<!ENTITY Upsilon     CDATA "&#933;" -- greek capital letter upsilon,  u+03A5 ISOgrk3 -->
<!ENTITY Phi      CDATA "&#934;" -- greek capital letter phi,  u+03A6 ISOgrk3 -->
<!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 ISOamso -->
<!ENTITY image    CDATA "&#8465;" -- blackletter capital I, =imaginary part, u+2111 ISOamso -->
<!ENTITY real     CDATA "&#8476;" -- blackletter capital R, =real part symbol, u+211C ISOamso -->
<!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 -->
<!-- Unicode 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 -->
<!-- Unicode 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 ISOamso -->
<!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 ISOamsb -->
<!-- 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 ISOamso -->
<!ENTITY and      CDATA "&#8869;" -- logical and, =wedge, u+2227 ISOtech -->
<!ENTITY or       CDATA "&#8870;" -- 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 sim      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 sube     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 -->
```

Named entities for markup-significant and internationalization characters

The named character entities 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 for details).

Entities have also been added for the remaining characters occuring in CP-1252 which do not occur in the HTMLlat1 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] or use other means. Display of glyphs for these characters may be obtained by being able to display the relevant [ISO10646] 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.

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 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 Unicode 2.0 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 OElig   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 Scaron  CDATA "&#352;"  -- latin capital letter s with caron, u+0160 ISOlat2 -->
<!ENTITY scaron  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 -->
<!ENTITY ensp    CDATA " "  -- en space, u+2002 ISOpub -->
<!ENTITY emsp    CDATA " "  -- em space, u+2003 ISOpub -->
<!ENTITY thinsp  CDATA " "  -- 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 Dagger  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 standardised -->
<!ENTITY rsaquo  CDATA "&#8250;"  -- single right-pointing angle quotation mark, u+203A ISO proposed -->
<!-- rsaquo is proposed but not yet ISO standardised -->
```

References

Contents

Normative references

[RFC822]
"Standard for the Format of ARPA Internet Text Messages", Revised by David H. Crocker, August 1982.
Download from ftp://ds.internic.net/rfc/rfc822.txt.

[RFC1123]
"Requirements for Internet Hosts -- Application and Support", R. Braden, October 1989.
Download from ftp://ds.internic.net/rfc/rfc1123.txt.

[RFC1468]
"Japanese Character Encoding for Internet Messages", J. Murai, M. Crispin, and E. van der Poel, June 1993.
Download from ftp://ds.internic.net/rfc/rfc1468.txt.

[RFC1555]
"Hebrew Character Encoding for Internet Messages", H. Nussbacher and Y. Bourvine, December 1993.
Download from ftp://ds.internic.net/rfc/rfc1555.txt.

[RFC1556]
"Handling of Bi-directional Texts in MIME", H. Nussbacher, December 1993.
Download from ftp://ds.internic.net/rfc/rfc1556.txt.

[RFC1590]
"Media Type Registration Procedure", J. Postel, March 1994.
Download from ftp://ds.internic.net/rfc/rfc1590.txt.

[RFC1738]
"Uniform Resource Locators", T. Berners-Lee, L. Masinter, and M. McCahill, December 1994.
Download from ftp://ds.internic.net/rfc/rfc1738.txt.

[RFC1766]
"Tags for the Identification of Languages", H. Alvestrand, March 1995.
Download from ftp://ds.internic.net/rfc/rfc1766.txt.

[RFC1808]
"Relative Uniform Resource Locators", R. Fielding, June 1995.
Download from ftp://ds.internic.net/rfc/rfc1808.txt.

[RFC1867]
"Form-based File Upload in HTML", E. Nebel and L. Masinter, November 1995.
Download from ftp://ds.internic.net/rfc/rfc1867.txt.

[RFC2045]
"Multipurpose Internet Mail Extensions (MIME) Part One: Format of Internet Message Bodies", N. Freed and N. Borenstein, November 1996.
Download from ftp://ds.internic.net/rfc/rfc2045.txt. 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.
Download from ftp://ds.internic.net/rfc/rfc2046.txt. Note that this RFC obsoletes RFC1521, RFC1522, and RFC1590.
[RFC2068]
"HTTP Version 1.1 ", R. Fielding, J. Gettys, J. Mogul, H. Frystyk Nielsen, and T. Berners-Lee, January 1997.
Download from ftp://ds.internic. net/rfc/rfc2068.txt.
[RFC2070]
"Internationalization of the HyperText Markup Language", F. Yergeau, G. Nicol, G. Adams, and M. Dürst, January 1997.
Download from ftp://ds.internic.net/rfc/rfc2070.txt.
[RFC2119]
"Key words for use in RFCs to Indicate Requirement Levels", S. Bradner, March 1997.
Download from ftp://ds.internic.net/rfc/rfc2119.txt.
[ISO639]
"Code for the representation of names of languages", ISO 639:1988.
For more information, consult http://www.iso.ch/cate/d4766.html.
See also http://www.sil.org/sgml/iso639a.html.
[ISO646]
"Information technology -- ISO 7-bit coded character set for information interchange, ISO/IEC 646:1991.
[ISO1000]
"SI units and recommendations for the user of their multiples and of certain other units", ISO 1000:1992.
[ISO3166]
"Codes for the representation of names of countries", ISO 3166:1993.
[ISO4217]
"Codes for the representation of currencies and funds", ISO 4217:1995.
[ISO8601]
"Data elements and interchange formats -- Information interchange -- Representation of dates and times", ISO 8601:1988.
[ISO88591]
"Information processing -- 8-bit single-byte coded graphic character sets -- Part 1: Latin alphabet No. 1", ISO 8859-1:1987.
[ISO8879]
"Information Processing -- Text and Office Systems -- Standard Generalized Markup Language (SGML)", ISO 8879:1986.
For the list of SGML entities, consult ftp://ftp.ifi.uio.no/pub/SGML/ENTITIES/.
[ISO10646]
"Information Technology - Universal Multiple- Octet Coded Character Set (UCS) - Part 1: Architecture and Basic Multilingual Plane", ISO/IEC 10646-1:1993. The current specification also takes into consideration the first five amendments to ISO/IEC 10646-1:1993.

[UNICODE]
"The Unicode Standard: Version 2.0", The Unicode Consortium, Addison-Wesley Developers Press, 1996.
For more information, consult the Unicode Consortium's home page at http://www.unicode.org/
[SRGB]
"A Standard Default color Space for the Internet", version 1.10, M. Stokes, M. Anderson, S. Chandrasekar, and R. Motta, 5 November 1996.
Download from http://www.w3.org/Graphics/Color/sRGB.html
[CSS1]
"Cascading Style Sheets, level 1", H. W. Lie and B. Bos, 17 December 1996.
Download from http://www.w3.org/TR/REC-CSS1-961217.html
[ADOBE90]
"Postscript Language Reference Manual", 2nd Edition, Appendix E., Addison-Wesley Publishing Co., 1990.
[ERCS]
"Extended Reference Concrete Syntax for SGML", 30 May 1995.
Download from http://www.sgmlopen.org/sgml/docs/ercs/ercs-home.html

Informative references

[MIMETYPES]
Download a list of registered Internet Media Types (MIME types) from ftp://ftp.isi.edu/in-notes/iana/assignments/media-types/.
[RFC850]
"Standard for Interchange of USENET Messages", M. Horton, June 1983.
Download from ftp://ds.internic.net/rfc/rfc850.txt.
[RFC1866]
"HyperText Markup Language 2.0", T. Berners-Lee and D. Connolly, November 1995.
Download from ftp://ds.internic.net/rfc/rfc1866.txt.
[HTML30]
"HyperText Markup Language Specification Version 3.0", Dave Raggett, September 1995.
Download from HyperText Markup Language Specification Version 3.0.
[HTML32]
"HTML 3.2 Reference Specification", Dave Raggett, 14 January 1997.
Download from http://www.w3.org/TR/REC-html32.html
[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 at at http://navysgml.dt.navy.mil/cals.html.
[RFC1942]
"HTML Tables", Dave Raggett, May 1996.
Download from ftp://ds.internic.net/rfc/rfc1942.txt.
[HTML3STYLE]
"HTML and Style Sheets", B. Bos, D. Raggett, and H. Lie, 24 March 1997.
Download from http://www.w3.org/TR/WD-style.html

[LEXHTML]
"A Lexical Analyzer for HTML and Basic SGML", D. Connolly, 15 June 1996. Download from http://www.w3.org/TR/WD-html-lex

[BRYAN88]
"SGML: An Author's Guide to the Standard Generalized Markup Language", M. Bryan, Addison-Wesley Publishing Co., 1988.

[GOLD90]
"The SGML Handbook", C. F. Goldfarb, Clarendon Press, 1991.

[VANH90]
"Practical SGML", E. van Herwijnen, Kluwer Academicq Publishers Group, Norwell and Dordrecht, 1990.

[SQ91]
"The SGML Primer", 3rd Edition, SoftQuad Inc., 1991.

[ETHNO]
"Ethnologue, Languages of the World", 12th Edition, Barbara F. Grimes editor, Summer Institute of Linguistics, October 1992.

[DCORE]
The Dublin Core: for more information see http://purl.org/metadata/dublin_core

[PICS]
Platform for Internet Content (PICS). For more information see http://www.w3.org/PICS/

[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.

Indexes

Contents This specification includes the following indexes for quick reference:

Index of Elements

*Legend: **R**equired / **O**ptional / **F**orbidden, **Y**es / **N**o*

Name	Start Tag	End Tag	Empty
A	R	R	N
ACRONYM	R	R	N
ADDRESS	R	R	N
APPLET	R	R	N
AREA	R	F	Y
B	R	R	N
BASE	R	F	Y
BASEFONT	R	F	Y
BDO	R	R	N
BIG	R	R	N
BLOCKQUOTE	R	R	N
BODY	O	O	N
BR	R	F	Y
BUTTON	R	R	N
CAPTION	R	R	N
CENTER	R	R	N
CITE	R	R	N
CODE	R	R	N
COL	R	F	Y
COLGROUP	R	O	N
DD	R	O	N
DEL	R	R	N
DFN	R	R	N
DIR	R	R	N

DIV	R	R	N
DL	R	R	N
DT	R	O	N
EM	R	R	N
FIELDSET	R	R	N
FONT	R	R	N
FORM	R	R	N
FRAME	R	F	Y
FRAMESET	R	R	N
H1	R	R	N
H2	R	R	N
H3	R	R	N
H4	R	R	N
H5	R	R	N
H6	R	R	N
HEAD	O	O	N
HR	R	F	Y
HTML	O	O	N
I	R	R	N
IFRAME	R	R	N
IMG	R	F	Y
INPUT	R	F	Y
INS	R	R	N
ISINDEX	R	F	Y
KBD	R	R	N
LABEL	R	R	N
LEGEND	R	R	N
LI	R	O	N

LINK	R	F	Y
MAP	R	R	N
MENU	R	R	N
META	R	F	Y
NOFRAMES	R	R	N
NOSCRIPT	R	R	N
OBJECT	R	R	N
OL	R	R	N
OPTION	R	O	N
P	R	O	N
PARAM	R	F	Y
PRE	R	R	N
Q	R	R	N
S	R	R	N
SAMP	R	R	N
SCRIPT	R	R	N
SELECT	R	R	N
SMALL	R	R	N
SPAN	R	R	N
STRIKE	R	R	N
STRONG	R	R	N
STYLE	R	R	N
SUB	R	R	N
SUP	R	R	N
TABLE	R	R	N
TBODY	O	O	N
TD	R	O	N
TEXTAREA	R	R	N

TFOOT	R	O	N
TH	R	O	N
THEAD	R	O	N
TITLE	R	R	N
TR	R	O	N
TT	R	R	N
U	R	R	N
UL	R	R	N
VAR	R	R	N

Index of Attributes

Name	Related Elements	Type	Default	Comment
accept-charset	FORM	CDATA	#IMPLIED	list of supported charsets
accept	INPUT	CDATA	#IMPLIED	list of MIME types for file upload
accesskey	A, LABEL, LEGEND	CDATA	#IMPLIED	accessibility key character
action	FORM	%URL	#REQUIRED	server-side form handler
align	CAPTION	%CAlign	#IMPLIED	relative to table
align	APPLET, IFRAME, IMG, INPUT	%IAlign	#IMPLIED	vertical or horizontal alignment
align	LEGEND	%LAlign	#IMPLIED	relative to fieldset
align	OBJECT	%OAlign	#IMPLIED	positioning inside document
align	TABLE	%TAlign	#IMPLIED	table position relative to window
align	DIV, H1, H2, H3, H4, H5, H6, P	(left | center | right | justify)	#IMPLIED	align, text alignment
align	COL, COLGROUP, TBODY, TD, TFOOT, TH, THEAD, TR	(left | center | right | justify | char)	#IMPLIED	
align	HR	(left | right | center)	#IMPLIED	
alink	BODY	CDATA	#IMPLIED	bgcolor, text, link, vlink, alink
alt	APPLET, IMG, INPUT	CDATA	#IMPLIED	description for text only browsers
alt	AREA	CDATA	#REQUIRED	description for text only browsers
archive	APPLET	CDATA	#IMPLIED	comma separated archive list
axes	TD, TH	CDATA	#IMPLIED	list of axis names

axis	TD, TH	CDATA	#IMPLIED	defaults to cell content
background	BODY	%URL	#IMPLIED	texture tile for document background
bgcolor	TABLE	%Color	#IMPLIED	background color for cells
bgcolor	TR	%Color	#IMPLIED	background color for row
bgcolor	TD, TH	%Color	#IMPLIED	cell background color
bgcolor	BODY	CDATA	#IMPLIED	
border	OBJECT	%Length	#IMPLIED	suggested link border width
border	IMG	%Pixels	#IMPLIED	suggested link border width
border	TABLE	CDATA	#IMPLIED	controls frame width around table
cellpadding	TABLE	CDATA	#IMPLIED	spacing within cells
cellspacing	TABLE	CDATA	#IMPLIED	spacing between cells
char	COL, COLGROUP, TBODY, TD, TFOOT, TH, THEAD, TR	CDATA	#IMPLIED	alignment char, e.g. char=':'
charoff	COL, COLGROUP, TBODY, TD, TFOOT, TH, THEAD, TR	CDATA	#IMPLIED	offset for alignment char
charset	A	CDATA	#IMPLIED	char encoding of linked resource
checked	INPUT	(checked)	#IMPLIED	for radio buttons and check boxes
cite	BLOCKQUOTE, Q	%URL	#IMPLIED	URL for source document or msg
cite	DEL, INS	%URL	#IMPLIED	info on reason for change

class	A, ACRONYM, ADDRESS, B, BIG, BLOCKQUOTE, BODY, BR, BUTTON, CAPTION, CENTER, CITE, CODE, COL, COLGROUP, DD, DEL, DFN, DIR, DIV, DL, DT, EM, FIELDSET, FORM, H1, H2, H3, H4, H5, H6, HR, I, IMG, INPUT, INS, ISINDEX, KBD, LABEL, LEGEND, LI, LINK, MAP, MENU, OBJECT, OL, OPTION, P, PRE, Q, S, SAMP, SELECT, SMALL, SPAN, STRIKE, STRONG, SUB, SUP, TABLE, TBODY, TD, TEXTAREA, TFOOT, TH, THEAD, TR, TT, U, UL, VAR	CDATA	#IMPLIED	space separated list of classes
classid	OBJECT	%URL	#IMPLIED	identifies an implementation
clear	BR	(left | all | right | none)	none	control of text flow
code	APPLET	CDATA	#IMPLIED	applet class file
codebase	APPLET	%URL	#IMPLIED	optional base URL for applet
codebase	OBJECT	%URL	#IMPLIED	some systems need an additional URL
codetype	OBJECT	%ContentType	#IMPLIED	Internet content type for code
color	BASEFONT, FONT	CDATA	#IMPLIED	#RRGGBB in hex, e.g. red: "#FF0000"

cols	FRAMESET	CDATA	#IMPLIED	if not given, default is 1 column
cols	TABLE	NUMBER	#IMPLIED	used for immediate display mode
cols	TEXTAREA	NUMBER	#REQUIRED	
colspan	TD, TH	NUMBER	1	number of cols spanned by cell
compact	DIR, MENU	(compact)	#IMPLIED	
compact	DL, OL, UL	(compact)	#IMPLIED	reduced interitem spacing
content	META	CDATA	#REQUIRED	associated information
coords	AREA	%Coords	#IMPLIED	comma separated list of values
coords	A	%Coords	#IMPLIED	for use with OBJECT SHAPES
data	OBJECT	%URL	#IMPLIED	reference to object's data
datetime	DEL, INS	CDATA	#IMPLIED	when changed: ISO date format
declare	OBJECT	(declare)	#IMPLIED	declare but don't instantiate flag

dir	A, ACRONYM, ADDRESS, B, BIG, BLOCKQUOTE, BODY, BUTTON, CAPTION, CENTER, CITE, CODE, COL, COLGROUP, DD, DEL, DFN, DIR, DIV, DL, DT, EM, FIELDSET, FORM, H1, H2, H3, H4, H5, H6, HEAD, HTML, I, IMG, INPUT, INS, ISINDEX, KBD, LABEL, LEGEND, LI, LINK, MENU, META, OBJECT, OL, OPTION, P, PRE, Q, S, SAMP, SELECT, SMALL, SPAN, STRIKE, STRONG, STYLE, SUB, SUP, TABLE, TBODY, TD, TEXTAREA, TFOOT, TH, THEAD, TITLE, TR, TT, U, UL, VAR	(ltr | rtl)	#IMPLIED	direction for weak/neutral text
dir	BDO	(ltr | rtl)	#REQUIRED	directionality
disabled	BUTTON, INPUT, LABEL, OPTION, SELECT, TEXTAREA	(disabled)	#IMPLIED	control is unavailable in this context
enctype	FORM	%ContentType	"application/x-www-form-urlencoded"	
face	BASEFONT, FONT	CDATA	#IMPLIED	comma separated list of font names
for	LABEL	IDREF	#IMPLIED	matches field ID value
frame	TABLE	%TFrame	#IMPLIED	which parts of table frame to include

frameborder	FRAME, IFRAME	(1 | 0)	1	request frame borders?
height	IFRAME, OBJECT	%Length	#IMPLIED	suggested height
height	IMG	%Pixels	#IMPLIED	suggested height in pixels
height	APPLET	%Pixels	#REQUIRED	suggested height in pixels
href	A, LINK	%URL	#IMPLIED	URL for linked resource
href	AREA	%URL	#IMPLIED	this region acts as hypertext link
href	BASE	%URL	#REQUIRED	
hspace	OBJECT	%Length	#IMPLIED	suggested horizontal gutter
hspace	APPLET, IMG	%Pixels	#IMPLIED	suggested horizontal gutter
http-equiv	META	NAME	#IMPLIED	HTTP response header name

id	A, ACRONYM, ADDRESS, B, BIG, BLOCKQUOTE, BODY, BR, BUTTON, CAPTION, CENTER, CITE, CODE, COL, COLGROUP, DD, DEL, DFN, DIR, DIV, DL, DT, EM, FIELDSET, FORM, H1, H2, H3, H4, H5, H6, HR, I, IMG, INPUT, INS, ISINDEX, KBD, LABEL, LEGEND, LI, LINK, MAP, MENU, OBJECT, OL, OPTION, P, PRE, Q, S, SAMP, SELECT, SMALL, SPAN, STRIKE, STRONG, SUB, SUP, TABLE, TBODY, TD, TEXTAREA, TFOOT, TH, THEAD, TR, TT, U, UL, VAR	ID	#IMPLIED	document-wide unique id
ismap	IMG	(ismap)	#IMPLIED	use server-side image map

lang	A, ACRONYM, ADDRESS, B, BDO, BIG, BLOCKQUOTE, BODY, BUTTON, CAPTION, CENTER, CITE, CODE, COL, COLGROUP, DD, DEL, DFN, DIR, DIV, DL, DT, EM, FIELDSET, FORM, H1, H2, H3, H4, H5, H6, HEAD, HTML, I, IMG, INPUT, INS, ISINDEX, KBD, LABEL, LEGEND, LI, LINK, MENU, META, OBJECT, OL, OPTION, P, PRE, Q, S, SAMP, SELECT, SMALL, SPAN, STRIKE, STRONG, STYLE, SUB, SUP, TABLE, TBODY, TD, TEXTAREA, TFOOT, TH, THEAD, TITLE, TR, TT, U, UL, VAR	NAME	#IMPLIED	[RFC1766] language value
language	SCRIPT	CDATA	#IMPLIED	predefined script language name
link	BODY	CDATA	#IMPLIED	
marginheight	FRAME, IFRAME	%Pixels	#IMPLIED	margin height in pixels
marginwidth	FRAME, IFRAME	%Pixels	#IMPLIED	margin widths in pixels
maxlength	INPUT	NUMBER	#IMPLIED	max chars for text fields
media	STYLE	CDATA	#IMPLIED	designed for use with these media
media	LINK	CDATA	#IMPLIED	for rendering on these media

method	FORM	(GET | POST)	GET	HTTP method used to submit the form
multiple	SELECT	(multiple)	#IMPLIED	default is single selection
name	OBJECT	%URL	#IMPLIED	submit as part of form
name	MAP	CDATA	#IMPLIED	
name	APPLET	CDATA	#IMPLIED	allows applets to find each other
name	BUTTON	CDATA	#IMPLIED	for scripting/forms as submit button
name	FRAME, IFRAME	CDATA	#IMPLIED	name of frame for targetting
name	A	CDATA	#IMPLIED	named link end
name	INPUT	CDATA	#IMPLIED	required for all but submit & reset
name	TEXTAREA	CDATA	#REQUIRED	
name	SELECT	CDATA	#REQUIRED	field name
name	PARAM	CDATA	#REQUIRED	property name
name	META	NAME	#IMPLIED	metainformation name
nohref	AREA	(nohref)	#IMPLIED	this region has no action
noresize	FRAME	(noresize)	#IMPLIED	allow users to resize frames?
noshade	HR	(noshade)	#IMPLIED	
nowrap	TD, TH	(nowrap)	#IMPLIED	suppress word wrap
object	APPLET	CDATA	#IMPLIED	serialized applet file
onblur	BUTTON, INPUT, LABEL, SELECT, TEXTAREA	%Script	#IMPLIED	the element lost the focus
onchange	INPUT, SELECT, TEXTAREA	%Script	#IMPLIED	the element value was changed

onclick	A, ACRONYM, ADDRESS, B, BIG, BLOCKQUOTE, BODY, BUTTON, CAPTION, CENTER, CITE, CODE, COL, COLGROUP, DD, DEL, DFN, DIR, DIV, DL, DT, EM, FIELDSET, FORM, H1, H2, H3, H4, H5, H6, HR, I, IMG, INPUT, INS, KBD, LABEL, LEGEND, LI, LINK, MENU, OBJECT, OL, OPTION, P, PRE, Q, S, SAMP, SELECT, SMALL, SPAN, STRIKE, STRONG, SUB, SUP, TABLE, TBODY, TD, TEXTAREA, TFOOT, TH, THEAD, TR, TT, U, UL, VAR	CDATA	#IMPLIED	the pointing device button was clicked

ondblclick	A, ACRONYM, ADDRESS, B, BIG, BLOCKQUOTE, BODY, BUTTON, CAPTION, CENTER, CITE, CODE, COL, COLGROUP, DD, DEL, DFN, DIR, DIV, DL, DT, EM, FIELDSET, FORM, H1, H2, H3, H4, H5, H6, HR, I, IMG, INPUT, INS, KBD, LABEL, LEGEND, LI, LINK, MENU, OBJECT, OL, OPTION, P, PRE, Q, S, SAMP, SELECT, SMALL, SPAN, STRIKE, STRONG, SUB, SUP, TABLE, TBODY, TD, TEXTAREA, TFOOT, TH, THEAD, TR, TT, U, UL, VAR	CDATA	#IMPLIED	the pointing device button was double clicked
onfocus	BUTTON, INPUT, LABEL, SELECT, TEXTAREA	%Script	#IMPLIED	the element got the focus

onkeydown	A, ACRONYM, ADDRESS, B, BIG, BLOCKQUOTE, BODY, BUTTON, CAPTION, CENTER, CITE, CODE, COL, COLGROUP, DD, DEL, DFN, DIR, DIV, DL, DT, EM, FIELDSET, FORM, H1, H2, H3, H4, H5, H6, HR, I, IMG, INPUT, INS, KBD, LABEL, LEGEND, LI, LINK, MENU, OBJECT, OL, OPTION, P, PRE, Q, S, SAMP, SELECT, SMALL, SPAN, STRIKE, STRONG, SUB, SUP, TABLE, TBODY, TD, TEXTAREA, TFOOT, TH, THEAD, TR, TT, U, UL, VAR	CDATA	#IMPLIED	a key was pressed down

onkeypress	A, ACRONYM, ADDRESS, B, BIG, BLOCKQUOTE, BODY, BUTTON, CAPTION, CENTER, CITE, CODE, COL, COLGROUP, DD, DEL, DFN, DIR, DIV, DL, DT, EM, FIELDSET, FORM, H1, H2, H3, H4, H5, H6, HR, I, IMG, INPUT, INS, KBD, LABEL, LEGEND, LI, LINK, MENU, OBJECT, OL, OPTION, P, PRE, Q, S, SAMP, SELECT, SMALL, SPAN, STRIKE, STRONG, SUB, SUP, TABLE, TBODY, TD, TEXTAREA, TFOOT, TH, THEAD, TR, TT, U, UL, VAR	CDATA	#IMPLIED	a key was pressed and released

onkeyup	A, ACRONYM, ADDRESS, B, BIG, BLOCKQUOTE, BODY, BUTTON, CAPTION, CENTER, CITE, CODE, COL, COLGROUP, DD, DEL, DFN, DIR, DIV, DL, DT, EM, FIELDSET, FORM, H1, H2, H3, H4, H5, H6, HR, I, IMG, INPUT, INS, KBD, LABEL, LEGEND, LI, LINK, MENU, OBJECT, OL, OPTION, P, PRE, Q, S, SAMP, SELECT, SMALL, SPAN, STRIKE, STRONG, SUB, SUP, TABLE, TBODY, TD, TEXTAREA, TFOOT, TH, THEAD, TR, TT, U, UL, VAR	CDATA	#IMPLIED	a key was released
onload	FRAMESET	%Script	#IMPLIED	all the frames have been loaded
onload	BODY	%Script	#IMPLIED	the document has been loaded

onmousedown	A, ACRONYM, ADDRESS, B, BIG, BLOCKQUOTE, BODY, BUTTON, CAPTION, CENTER, CITE, CODE, COL, COLGROUP, DD, DEL, DFN, DIR, DIV, DL, DT, EM, FIELDSET, FORM, H1, H2, H3, H4, H5, H6, HR, I, IMG, INPUT, INS, KBD, LABEL, LEGEND, LI, LINK, MENU, OBJECT, OL, OPTION, P, PRE, Q, S, SAMP, SELECT, SMALL, SPAN, STRIKE, STRONG, SUB, SUP, TABLE, TBODY, TD, TEXTAREA, TFOOT, TH, THEAD, TR, TT, U, UL, VAR	CDATA	#IMPLIED	the pointing device button was pressed down

onmousemove	A, ACRONYM, ADDRESS, B, BIG, BLOCKQUOTE, BODY, BUTTON, CAPTION, CENTER, CITE, CODE, COL, COLGROUP, DD, DEL, DFN, DIR, DIV, DL, DT, EM, FIELDSET, FORM, H1, H2, H3, H4, H5, H6, HR, I, IMG, INPUT, INS, KBD, LABEL, LEGEND, LI, LINK, MENU, OBJECT, OL, OPTION, P, PRE, Q, S, SAMP, SELECT, SMALL, SPAN, STRIKE, STRONG, SUB, SUP, TABLE, TBODY, TD, TEXTAREA, TFOOT, TH, THEAD, TR, TT, U, UL, VAR	CDATA	#IMPLIED	the pointing device was moved

onmouseout	A, ACRONYM, ADDRESS, B, BIG, BLOCKQUOTE, BODY, BUTTON, CAPTION, CENTER, CITE, CODE, COL, COLGROUP, DD, DEL, DFN, DIR, DIV, DL, DT, EM, FIELDSET, FORM, H1, H2, H3, H4, H5, H6, HR, I, IMG, INPUT, INS, KBD, LABEL, LEGEND, LI, LINK, MENU, OBJECT, OL, OPTION, P, PRE, Q, S, SAMP, SELECT, SMALL, SPAN, STRIKE, STRONG, SUB, SUP, TABLE, TBODY, TD, TEXTAREA, TFOOT, TH, THEAD, TR, TT, U, UL, VAR	CDATA	#IMPLIED	the pointing device was moved away

onmouseover	A, ACRONYM, ADDRESS, B, BIG, BLOCKQUOTE, BODY, BUTTON, CAPTION, CENTER, CITE, CODE, COL, COLGROUP, DD, DEL, DFN, DIR, DIV, DL, DT, EM, FIELDSET, FORM, H1, H2, H3, H4, H5, H6, HR, I, IMG, INPUT, INS, KBD, LABEL, LEGEND, LI, LINK, MENU, OBJECT, OL, OPTION, P, PRE, Q, S, SAMP, SELECT, SMALL, SPAN, STRIKE, STRONG, SUB, SUP, TABLE, TBODY, TD, TEXTAREA, TFOOT, TH, THEAD, TR, TT, U, UL, VAR	CDATA	#IMPLIED	the pointing device was moved over

onmouseup	A, ACRONYM, ADDRESS, B, BIG, BLOCKQUOTE, BODY, BUTTON, CAPTION, CENTER, CITE, CODE, COL, COLGROUP, DD, DEL, DFN, DIR, DIV, DL, DT, EM, FIELDSET, FORM, H1, H2, H3, H4, H5, H6, HR, I, IMG, INPUT, INS, KBD, LABEL, LEGEND, LI, LINK, MENU, OBJECT, OL, OPTION, P, PRE, Q, S, SAMP, SELECT, SMALL, SPAN, STRIKE, STRONG, SUB, SUP, TABLE, TBODY, TD, TEXTAREA, TFOOT, TH, THEAD, TR, TT, U, UL, VAR	CDATA	#IMPLIED	the pointing device button was released
onreset	FORM	%Script	#IMPLIED	the form was reset
onselect	INPUT, SELECT, TEXTAREA	%Script	#IMPLIED	some text was selected
onsubmit	FORM	%Script	#IMPLIED	the form was submitted
onunload	FRAMESET	%Script	#IMPLIED	all the frames have been removed
onunload	BODY	%Script	#IMPLIED	the document has been removed
profile	HEAD	%URL	#IMPLIED	named dictionary of meta info
prompt	ISINDEX	CDATA	#IMPLIED	prompt message
readonly	TEXTAREA	(readonly)	#IMPLIED	

readonly	INPUT	(readonly)	#IMPLIED	for text and passwd
rel	A, LINK	CDATA	#IMPLIED	forward link types
rev	A, LINK	CDATA	#IMPLIED	reverse link types
rows	FRAMESET	CDATA	#IMPLIED	if not given, default is 1 row
rows	TEXTAREA	NUMBER	#REQUIRED	
rowspan	TD, TH	NUMBER	1	number of rows spanned by cell
rules	TABLE	%TRules	#IMPLIED	rulings between rows and cols
scheme	META	CDATA	#IMPLIED	select form of content
scrolling	FRAME, IFRAME	(yes | no | auto)	auto	scrollbar or none
selected	OPTION	(selected)	#IMPLIED	
shape	AREA	%Shape	rect	controls interpretation of coords
shape	A	%Shape	rect	for use with OBJECT SHAPES
shapes	OBJECT	(shapes)	#IMPLIED	object has shaped hypertext links
size	HR	%Pixels	#IMPLIED	
size	FONT	CDATA	#IMPLIED	[+]nn e.g. size="+1", size=4
size	INPUT	CDATA	#IMPLIED	specific to each type of field
size	BASEFONT	CDATA	#REQUIRED	base font size for FONT elements
size	SELECT	NUMBER	#IMPLIED	rows visible
span	COLGROUP	NUMBER	1	default number of columns in group
span	COL	NUMBER	1	number of columns spanned by group
src	SCRIPT	%URL	#IMPLIED	URL for an external script
src	INPUT	%URL	#IMPLIED	for fields with images

src	FRAME, IFRAME	%URL	#IMPLIED	source of frame content
src	IMG	%URL	#REQUIRED	URL of image to embed
standby	OBJECT	CDATA	#IMPLIED	message to show while loading
start	OL	NUMBER	#IMPLIED	starting sequence number
style	A, ACRONYM, ADDRESS, B, BIG, BLOCKQUOTE, BODY, BR, BUTTON, CAPTION, CENTER, CITE, CODE, COL, COLGROUP, DD, DEL, DFN, DIR, DIV, DL, DT, EM, FIELDSET, FORM, H1, H2, H3, H4, H5, H6, HR, I, IMG, INPUT, INS, ISINDEX, KBD, LABEL, LEGEND, LI, LINK, MAP, MENU, OBJECT, OL, OPTION, P, PRE, Q, S, SAMP, SELECT, SMALL, SPAN, STRIKE, STRONG, SUB, SUP, TABLE, TBODY, TD, TEXTAREA, TFOOT, TH, THEAD, TR, TT, U, UL, VAR	CDATA	#IMPLIED	associated style info
tabindex	A, AREA, BUTTON, INPUT, OBJECT, SELECT, TEXTAREA	NUMBER	#IMPLIED	position in tabbing order
target	AREA, BASE, LINK	CDATA	#IMPLIED	where to render linked resource

target	A	CDATA	#IMPLIED	where to render resource
target	FORM	CDATA	#IMPLIED	where to render result
text	BODY	CDATA	#IMPLIED	
title	STYLE	CDATA	#IMPLIED	advisory title
title	A, ACRONYM, ADDRESS, B, BIG, BLOCKQUOTE, BODY, BR, BUTTON, CAPTION, CENTER, CITE, CODE, COL, COLGROUP, DD, DEL, DFN, DIR, DIV, DL, DT, EM, FIELDSET, FORM, H1, H2, H3, H4, H5, H6, HR, I, IMG, INPUT, INS, ISINDEX, KBD, LABEL, LEGEND, LI, LINK, MAP, MENU, OBJECT, OL, OPTION, P, PRE, Q, S, SAMP, SELECT, SMALL, SPAN, STRIKE, STRONG, SUB, SUP, TABLE, TBODY, TD, TEXTAREA, TFOOT, TH, THEAD, TR, TT, U, UL, VAR	CDATA	#IMPLIED	advisory title/amplification
type	OBJECT	%ContentType	#IMPLIED	Internet content type for data
type	LINK	%ContentType	#IMPLIED	advisory Internet content type
type	INPUT	%InputType	TEXT	what kind of widget is needed
type	LI	%LIStyle	#IMPLIED	list item style
type	OL	%OLStyle	#IMPLIED	numbering style

type	UL	(%ULStyle)	#IMPLIED	bullet style
type	BUTTON	(submit | reset)	#IMPLIED	for use as form submit/reset button
type	SCRIPT	CDATA	#IMPLIED	Internet content type for script language
type	PARAM	CDATA	#IMPLIED	Internet media type
type	STYLE	CDATA	#REQUIRED	Internet content type for style language
usemap	OBJECT	%URL	#IMPLIED	reference to image map
usemap	IMG, INPUT	%URL	#IMPLIED	use client-side image map
valign	COL, COLGROUP, TBODY, TD, TFOOT, TH, THEAD, TR	(top | middle | bottom | baseline)	#IMPLIED	vertical alignment in cells
value	OPTION	CDATA	#IMPLIED	defaults to element content
value	BUTTON	CDATA	#IMPLIED	gets passed to server when submitted
value	PARAM	CDATA	#IMPLIED	property value
value	INPUT	CDATA	#IMPLIED	required for radio and checkboxes
value	LI	NUMBER	#IMPLIED	reset sequence number
valuetype	PARAM	(DATA | REF | OBJECT)	DATA	How to interpret value
version	HTML	CDATA	'http://www.w3.org/TR/WD-html40-970708/HTML4.dtd'	Constant
vlink	BODY	CDATA	#IMPLIED	
vspace	OBJECT	%Length	#IMPLIED	suggested vertical gutter
vspace	APPLET, IMG	%Pixels	#IMPLIED	suggested vertical gutter
width	HR	%Length	#IMPLIED	
width	IFRAME, OBJECT	%Length	#IMPLIED	suggested width
width	IMG	%Pixels	#IMPLIED	suggested width in pixels

width	APPLET	%Pixels	#REQUIRED	suggested width in pixels
width	COL	CDATA	#IMPLIED	column width specification
width	COLGROUP	CDATA	#IMPLIED	default width for enclosed COLs
width	TABLE	CDATA	#IMPLIED	table width relative to window
width	PRE	NUMBER	#IMPLIED	

Appendixes

Contents

Changes between HTML 3.2 and HTML 4.0

Contents

Changes to elements

New elements

The new elements in this version of HTML are `Q`, `INS`, `DEL`, `ACRONYM`, `LEGEND`, `COLGROUP`, `BUTTON`, and `FIELDSET`.

Deprecated elements

The following elements are now deprecated: `ISINDEX`, `APPLET`, `CENTER`, `FONT`, `BASEFONT`, `STRIKE`, `S`, `U`, `DIR`, and `MENU`.

Obsolete elements

The following elements are now obsolete: `XMP`, `PLAINTEXT`, and `LISTING`. For all of them, you should use the `PRE` element instead.

Changes to Tables

The HTML 4.0 table model has grown out of early work on HTML+ and the initial draft of HTML3.0. The earlier model has been been extended in response to requests from information providers for improved control over the presentation of tabular information:

- The ability to align on designated characters such as "." and":"
  (e.g., aligning a column of numbers on the decimal point).
- The need for more flexibility in specifying table frames and rules.
- The need for incremental display of large tables as data is received.
- The ability to support scrollable tables with fixed headers plus better support for breaking tables across pages for printing.
- The need for optional column based defaults for alignment properties

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 attribute compatible with the latest versions of the most popular browsers. Some clarifications have been made to the role of the `dir` attribute 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"` 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 `frame` and `rules` attributes with other table elements.

Changes to Forms

The forms specified in HTML 3.2 have the following problems:

- There is no provision for keyboard shortcuts for particular actions, for access keys for driving menus, etc.
- Although form controls can be made insensitive dynamically, they cannot be declared as such at initialization time.
- Along the same line form controls, such as form fields, cannot be made "read only".
- Labels for radio buttons and checkboxes are not sensitive, i.e., clicking on a label text doesn't effect the button state.
- There is no way to markup groups of related form fields in a way that effectively supports browsing with speech-based user agents.
- There is no provision for checking values as they are entered into form fields. All checking is done at the server when the form's contents are submitted.
- Nothing is provided to specify what type of data file is expected when the user is asked to submit files.
- Forms can only contain the two buttons submit and reset.
- There is no way to specify what character sets the server issuing a form can handle.

To solve these problems this specification introduces several new attributes and elements.

- The `accesskey` attribute provides for specifying direct keyboard access to form fields.
- The `disabled` attribute allows form providers to make a form control initially insensitive.
- And with the additional attribute `readonly`, authors can prohibit changes to a form field.
- 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, including onchange-INPUT, in association with support for scripting languages, allows form providers to verify user-entered data.
- The INPUT element has a new attribute accept that allows authors to specify a list of valid media types or type patterns for the input.
- The new BUTTON element can be used to make richer forms with more than just a submit and a reset button.
- The FORM element includes the attribute accept-charset, modeled on the HTTP "Accept-Charset" header (see [RFC2068]). This attribute (first proposed in [RFC1867]) may be used to specify a list of character sets acceptable to the server.

Performance, Implementation, and Design Notes

Contents

The following notes are informative, not normative.

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.

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" href="mydoc-fr.html"
         lang="fr" title="La vie souterrainne">
<LINK rel="alternate" href="mydoc-de.html"
         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 attribute is not defined by this specification. Consider these examples,

```
<META name="keywords" content="vacation,Greece,sunshine">
<META name="description" content="Idylic 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="begin"** along with a TITLE, as in:

```
<LINK rel="begin"
        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.

Search robots

The robots.txt file

When a Robot vists 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" URL 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 URL	URL 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: URL's are case-sensitive, and "/robots.txt" string must be all lower-case. Blank lines are not permitted.

There must be exactly one "User-agent" field. 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 URL that is not to be visited. This can be a full path, or a partial path; any URL 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 URLs 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`. The name and the content attribute values are case-insensitive.

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.

Notes on tables

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-TABLE` 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. It makes more sense to set the `cols` attribute to the number of columns rather than using some "layout" attribute (e.g., layout="fixed" or layout="auto").

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 appearence 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]), 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.

Recommended Layout Algorithms

If the `cols` attribute on the `TABLE` element specifies the number of columns, then 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.

We 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.

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 columns specified by the `cols` attribute may be inconsistent with the number of columns implied by the `COL` elements. This in turn, 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 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 `COLS` attribute is missing from the table start tag, 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 occuring 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 un-aligned. 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 (as used by Arena) 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 (i.e., englobing) 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 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-COLGROUP` attributes. This provides a measure of future proofing, as it is anticipated that 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.*

Notes on styles

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.

New media types

It is likely that the list of media types will grow in the future. To enable such extensions to be introduced smoothly, user agents conforming to this specification must be able to parse the media type attribute value as follows:

1. Comma characters (Unicode decimal 44) are used to break the media attribute value into a list of entries. For example,

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

   is mapped to:

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

2. Each entry is truncated just before the first character that isn't a *US ASCII letter* [a-zA-Z] (Unicode decimal 65-90, 97-122), or *hyphen* (Unicode decimal 45). In our example, this gives:

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

3. A case-insensitive match is then made with the set of media types defined above. Entries that don't match should be ignored. In the example we are left with `screen` and `print`.

Note: Style sheets may include media dependent variations within them. For instance the proposed CSS @media *construct. In such cases it may be appropriate to use the default value "all".*

Notes on forms

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 stil 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.

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.

Notes on scripting

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 instrinsic 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 reoccur when the document is resized, repainted, etc.

Here are some examples using JavaScript. The first one randomizes the document background color:

```
<BODY bgcolor='&{randomrbg()};'>
```

Perhaps you want to dim the background for evening viewing:

```
<BACKGROUND src='&{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(imageurl)};" href="&{myurl};">
 </MAP>
```

This example sets the size of an image based upon document properties:

```
<IMG src="bar.gif" width='&{document.banner.width/2};' height='50%'>
```

You can set the URL for a link or image by script:

```
 <SCRIPT>
   function manufacturer(widget) {
       ...
   }
   function location(manufacturer) {
       ...
   }
   function logo(manufacturer) {
       ...
   }
 </SCRIPT>
  <A href='&{location(manufacturer("widget"))};'>widget</A>
  <IMG src='&{logo(manufacturer("widget"))};'>
```

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;))};">
```

HTML and Organizations (W3C, IETF, ISO)

Contents

HTML 4.0 Specification

Table of Contents