Network Working Group
Request for Comments: 1942
Category: Experimental

D. Raggett
W3C

May 1996

HTML Tables

Status of this Memo

This memo defines an Experimental Protocol for the Internet community. This memo does not specify an Internet standard of any kind. Discussion and suggestions for improvement are requested. Distribution of this memo is unlimited.

Abstract

The HyperText Markup Language (HTML) is a simple markup language used to create hypertext documents that are portable from one platform to another. HTML documents are SGML documents with generic semantics that are appropriate for representing information from a wide range of applications. This specification extends HTML to support a wide variety of tables. The model is designed to work well with associated style sheets, but does not require them. It also supports rendering to braille, or speech, and exchange of tabular data with databases and spreadsheets. The HTML table model embodies certain aspects of the CALS table model, e.g. the ability to group table rows into thead, tbody and tfoot sections, plus the ability to specify cell alignment compactly for sets of cells according to the context.

Table of Contents

Recent Changes

This specification extends HTML to support tables. The table model has grown out of early work on HTML+ and the initial draft of HTML3. The earlier model has been been extended in response to requests from information providers for improved control over the presentation of tabular information:

* alignment on designated characters such as "." and ":"
  e.g. aligning a column of numbers on the decimal point

* more flexibility in specifying table frames and rules

* incremental display for 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

* 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. A subsidiary goal has been to simplify importing tables conforming to the SGML CALS model. The latest draft makes the ALIGN attribute compatible with the latest Netscape and Microsoft browsers. Some clarifications have been made to the role of the DIR attribute and recommended behaviour when absolute and relative column widths are mixed.

A new element COLGROUP has been introduced to allow sets of columns 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 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.

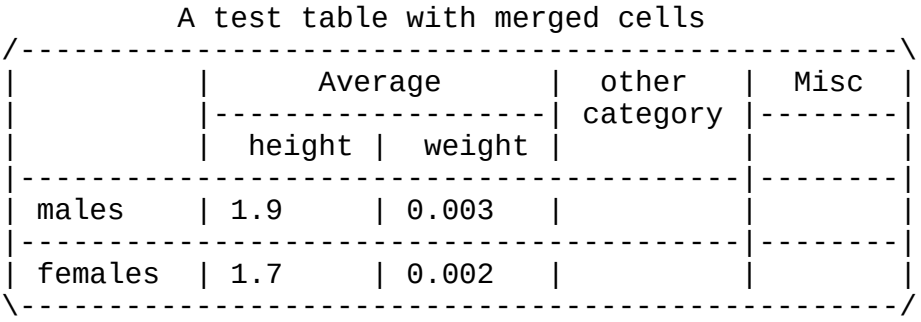
A Brief Introduction to HTML Tables

Tables start with an optional caption followed by one or more rows. Each row is formed by one or more cells, which are differentiated into header and data cells. Cells can be merged across rows and columns, and include attributes assisting rendering to speech and braille, or for exporting table data into databases. The model provides limited support for control over appearence, for example horizontal and vertical alignment of cell contents, border styles and cell margins. You can further affect this by grouping rows and columns together. Tables can contain a wide range of content, such as headers, lists, paragraphs, forms, figures, preformatted text and even nested tables.

Example

```
<TABLE 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>
```

On a dumb terminal, this would be rendered something like:



Next, a richer example with grouped rows and columns (adapted from "Developing International Software" by Nadine Kano). First here is what the table looks like on paper:

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 East. Europe	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

The markup for this uses COLGROUP elements to group columns and to set default column alignment. TBODY elements are used to group rows. The FRAME and RULES attributes are used to select which borders to render.

```
<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 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>
```

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 looking at a wide range of tabular layout in magazines, books and other paper-based documents. The model was chosen to allow simple tables to be expressed simply with extra complexity only 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 using filters from other document formats or direct wysiwyg editors for HTML. It is important that the HTML table model fits well with these routes for authoring HTML. This affects how the representation handles cells which span multiple rows or columns, and how alignment and other presentation properties are associated with groups of cells.

A major consideration for the HTML table model is that the fonts and window sizes etc. in use with browsers are not under the author’s control. This makes it risky to rely on column widths specified in terms of absolute units such as picas or pixels. Instead, tables can be dynamically sized to match the current window size and fonts. Authors can provide guidance as to the relative widths of columns, but user agents should to ensure that columns are wide enough to render the width of the largest single element of the cell’s content. If the author’s specification must be overridden, it is preferred that the relative widths of individual columns are not changed drastically.

For large tables or slow network connections, it is desirable to be able to start displaying the 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. Authors can provide a hint to user agents to activate incremental display of table contents. This feature requires the author to specify the number of columns, and includes provision for control of table width and the widths of different columns in relative or absolute terms.

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 including a WIDTH attribute in the TABLE start tag. 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 start tag.

Authors still need a way of informing the browser whether to use incremental display or to automatically size the table to match the cell contents. For the two pass auto sizing mode, the number of columns is determined by the first pass, while for the incremental mode, the number of columns needs to be stated up front. So it seems to that COLS=_nn_ would be better for this purpose than a LAYOUT attribute such as LAYOUT=FIXED or LAYOUT=AUTO.

It is generally held useful to consider documents from two perspectives: Structural idioms such as headers, paragraphs, lists, tables, and figures; and rendering idioms such as margins, leading, font names and sizes. The wisdom of past experience encourages us to separate the structural information in documents from rendering information. Mixing them together ends up causing increased cost of ownership for maintaining documents, and reduced portability between applications and media.

For tables, the alignment of text within table cells, and the borders between cells are, from the purist’s point of view, rendering information. 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 is designed to take advantage of such style sheets but not to require

them.

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, 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.

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.

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.

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.

A walk through the table DTD

The table document type definition provides the formal definition of the allowed syntax for html tables. The following is an annotated listing of the DTD. The complete listing appears at the end of this document.

Note that the TABLE element is a block-like element rather a character-level element. As such it is a peer of other HTML block-like elements such as paragraphs, lists and headers.

Common Attributes

The following attributes occur in several of the elements and are defined here for brevity. In general, all attribute names and values in this specification are case insensitive, except where noted otherwise. The ID, CLASS and attributes are required for use with style sheets, while LANG and DIR are needed for internationalization.

```
<!ENTITY % attrs
      "id      ID       #IMPLIED  -- element identifier --
       class   NAMES    #IMPLIED  -- for subclassing elements --
       lang    NAME     #IMPLIED  -- as per RFC 1766 --
       dir   (ltr|rtl)  #IMPLIED  -- I18N text direction --">
```

ID
Used to define a document-wide identifier. This can be used for naming positions within documents as the destination of a hypertext link. It may also be used by style sheets for rendering an element in a unique style. An ID attribute value is an SGML NAME token. NAME tokens are formed by an initial letter followed by letters, digits, "-" and "." characters. The letters are restricted to A-Z and a-z.

CLASS
A space separated list of SGML NAME tokens. CLASS names specify that the element belongs to the corresponding named classes. It allows authors to distinguish different roles played by the same tag. The classes may be used by style sheets to provide different renderings as appropriate to these roles.

LANG
A LANG attribute identifies the natural language used by the content of the associated element.The syntax and registry of language values are defined by RFC 1766. In summary the language is given as a primary tag followed by zero or more subtags, separated by "-". White space is not allowed and all tags are case insensitive. The name space of tags is administered by

IANA. The two letter primary tag is an ISO 639 language abbreviation, while the initial subtag is a two letter ISO 3166 country code. Example values for LANG include:

en, en-US, en-uk, i-cherokee, x-pig-latin.

DIR

Human writing systems are grouped into scripts, which determine amongst other things, the direction the characters are written. Elements of the Latin script are nominally left to right, while those of the Arabic script are nominally right to left. These characters have what is called strong directionality. Other characters can be directionally neutral (spaces) or weak (punctuation).

The DIR attribute specifies an encapsulation boundary which governs the interpretation of neutral and weakly directional characters. It does not override the directionality of strongly directional characters. The DIR attribute value is one of LTR for left to right, or RTL for right to left, e.g. DIR=RTL.

When applied to TABLE, it indicates the geometric layout of rows (i.e. row 1 is on right if DIR=RTL, but on the left if DIR=LTR) and it indicates a default base directionality for any text in the table's content if no other DIR attribute applies to that text.

Horizontal and Vertical Alignment Attributes

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 element itself.

ALIGN

This specifies the horizontal alignment of cell contents.

```
<!-- horizontal alignment attributes for cell contents -->
<!ENTITY % cell.halign
        "align  (left|center|right|justify|char) #IMPLIED
         char    CDATA   #IMPLIED -- alignment char, e.g. char=':' --
         charoff CDATA   #IMPLIED -- offset for alignment char --"
        >
```

The attribute value should be one of LEFT, CENTER, RIGHT, JUSTIFY and CHAR. User agents may treat JUSTIFY as left alignment if they lack support for text justification. ALIGN=CHAR is used for aligning cell contents on a particular character.

For cells spanning multiple rows or columns, where the alignment property is inherited from the row or column, the initial row and column for the cell determines the appropriate alignment property to use.

Note that an alignment attribute on elements within the cell, e.g. on a P element, overrides the normal alignment value for the cell.

CHAR
: This is used to specify an alignment character for use with align=char, e.g. char=":". The default character is the decimal point for the current language, as set by the LANG attribute. The CHAR attribute value is case sensitive.

CHAROFF
: 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. The resolved direction of the cell, as determined by the inheritance of the DIR attribute, is used to set whether the offset is from the left or right margin of the cell. For Latin scripts, the offset will be from the left margin, while for Arabic scripts, it will be from the right margin. In addition to standard units, the "%" sign may be used to indicate that the value specifies the alignment position as a percentage offset of the current cell, e.g. CHAROFF="30%" indicates the alignment character should be positioned 30% through the cell.

  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 width 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.

VALIGN
: Defines whether the cell contents are aligned with the top, middle or bottom of the cell.

```
<!-- vertical alignment attributes for cell contents -->
<!ENTITY % cell.valign
        "valign  (top|middle|bottom|baseline)  #IMPLIED"
        >
```

If present, the value of the attribute should be one of: TOP, MIDDLE, BOTTOM or BASELINE. All cells in the same row with valign=baseline should be vertically positioned so that the first text line in each such cell occur on a common baseline. This constraint does not apply to subsequent text lines in these cells.

Inheritance Order

Alignment properties can be included with most of the table elements: COL, THEAD, TBODY, TFOOT, TR, TH and TD. When rendering cells, horizontal alignment is determined by columns in preference to rows, while for vertical alignment, the rows are more important than the columns. The following table gives the detailed precedence order for each attribute, where X > Y denotes that X takes precedence over Y:

ALIGN, CHAR and CHAROFF:

cells > columns > column groups > rows > row groups > default

VALIGN, LANG, and DIR:

cells > rows > row groups > columns > column groups > table > default

Where cells are defined by TH and TD elements; rows by TR elements; row groups by THEAD, TBODY and TFOOT elements, columns by COL elements; and column groups by COLGROUP and COL elements. Note that there is no inheritance mechanism for the CLASS attribute.

Properties defined on cells take precedence over inherited properties, but are in turn over-ridden by alignment properties on elements within cells. In the absence of an ALIGN attribute along the inheritance path, the recommended default alignment for table cell contents is ALIGN=LEFT for table data and ALIGN=CENTER for table headers. The recommended default for vertical alignment is VALIGN=MIDDLE. These defaults are chosen to match the behaviour of the widely deployed Netscape implementation.

Standard Units for Widths

Several attributes specify widths as a number followed by an optional suffix. The units for widths are specified by the suffix: pt denotes points, pi denotes picas, in denotes inches, cm denotes centimeters,

mm denotes millimeters, em denotes em units (equal to the height of the default font), and px denotes screen pixels. The default units are screen pixels (chosen for backwards compatibility). The number is an integer value or a real valued number such as "2.5". Exponents, as in "1.2e2", are not allowed. White space is not allowed between the number and the suffix.

The above set of suffices is augmented for certain elements: "%" is used for the WIDTH attribute for the TABLE element. It indicates that the attribute specifies the percentage width of the space between the current left and right margins, e.g. width="50%". For the COL element, "*" is used with the WIDTH attribute to specify relative column widths, e.g. width="3*", using the same representation as the CALS table model.

The TABLE element

```
<!ENTITY % Where "(left|center|right)">

<!ELEMENT table - - (caption?, (col*|colgroup*), thead?, tfoot?, tbody+)>

<!ATTLIST table                     -- table element --
        %attrs;                     -- id, lang, dir and class --
        align   %Where;  #IMPLIED   -- table position relative to --
                                    -- window --
        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   %Frame;  #IMPLIED   -- which parts of table frame to --
                                    -- include --
        rules   %Rules;  #IMPLIED   -- controls rules between cells --
        cellspacing CDATA #IMPLIED -- spacing between cells --
        cellpadding CDATA #IMPLIED -- spacing within cells --
        >
```

The TABLE element requires both start and end tags. Table elements start with an optional CAPTION element, optionally followed by either one or more COL elements, or one or more COLGROUP elements, then an optional THEAD, an optional TFOOT, and finally one or more TBODY elements.

ID, CLASS, LANG and DIR
: See earlier description of common attributes.

ALIGN
: Defines the horizontal position of the table relative to the current left and right margins. ALIGN=CENTER centers the table

midway between the left and right margins. ALIGN=LEFT positions the table at the left margin, while ALIGN=RIGHT positions the table at the right margin. User agents may flow text around the right handside of the table for ALIGN=LEFT, or the left handside for ALIGN=RIGHT.

Note you can use <BR CLEAR=LEFT> after the table element if you want to avoid text flowing along side the table when you have specified ALIGN=LEFT, or <BR CLEAR=RIGHT> for a right aligned table. To prevent a right aligned table flowing around something else, use <BR CLEAR=RIGHT> before the table etc. Greater control over textflow is possible using style sheets.

WIDTH

Specifies the desired width of the table. In addition to the standard units, the "%" sign may used to indicate that the width specifies the percentage width of the space between the current left and right margins, e.g. width="50%". In the absence of this attribute, the table width can be determined by the layout algorithm given later on.

It is recommended that the table width be increased beyond the value indicated by the WIDTH attribute as needed to avoid any overflow of cell contents. Such increases should try to avoid drastic changes to relative column widths specified by the author. To avoid the need for excessive horizontal scrolling, or when such scrolling is impractical or undesired, it may be appropriate to split words across lines.

COLS

Specifies the number of columns for the table. If present the user agent may render the table dynamically as data is received from the network without waiting for the complete table to be received. If the WIDTH attribute is missing, a default of "100%" may be assumed for this purpose. If the COLS attribute is absent, a prepass through the table’s contents is needed to determine the number of columns together with suitable values for the widths of each column.

BORDER

Specifies the width of the border framing the table, see standard units.

FRAME

Specifies which sides of the frame to render.

```
<!ENTITY % Frame
   "(void|above|below|hsides|lhs|rhs|vsides|box|border)">
```

VOID
Don’t render any sides of the frame.

ABOVE
The top side of the frame

BELOW
The bottom side of the frame

HSIDES
The top and bottom sides of the frame

LHS
The left hand side of the frame

RHS
The right hand side of the frame

VSIDES
The left and right sides of the frame

BOX
All four sides of the frame

BORDER
All four sides of the frame

The value "Border" is included for backwards compatibility with deployed browsers. If a document includes <TABLE BORDER> the user agent will see FRAME=BORDER and BORDER=_implied_. If the document includes <TABLE BORDER=_n_> then the user agent should treat this as FRAME=BORDER except if _n=0_ for which FRAME=VOID is appropriate.

Note: it would have been preferable to choose values for FRAME 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 FRAME have been chosen to avoid clashes with the RULES, ALIGN and VALIGN attributes. This provides a measure of future proofing, as it is anticipated that 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 HTML-WG 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.

RULES
Specifies where to draw rules within the table interior.

<!ENTITY % Rules "(none | groups | rows | cols | all)">

NONE
Suppresses internal rulings.

GROUPS
The THEAD, TFOOT and TBODY elements divide the table into groups of rows, while COLGROUP elements divide the table into groups of columns. This choice places a horizontal rule between each row group and a vertical rule between each column group. Note that every table has at least one row and one column group.

ROWS
As RULES=GROUPS plus horizontal rules between all rows. User agents may choose to use a heavier rule between groups of rows and columns for emphasis.

COLS
As RULES=GROUPS plus vertical rules between all columns. User agents may choose to use a heavier rule between groups of rows and columns for emphasis.

ALL
Place rules between all rows and all columns. User agents may choose to use a heavier rule between groups of rows and columns for emphasis.

If a document includes <TABLE BORDER> or <TABLE BORDER=_n_> then the default for the table element is RULES=ALL, except if _n=0_ for which RULES=NONE is appropriate.

CELLSPACING
This attribute is intended for backwards compatibility with deployed user agents. It specifies the space between the table frame and the first or last cell border for each row or column, and between other cells in the table. See standard units. Greater control will be possible using style sheet languages.

CELLPADDING
This attribute is intended for backwards compatibility with deployed user agents. It specifies the amount of space between the border of the cell and its contents both above/below, and

left//right. See standard units. Greater control will be possible using style sheet languages.

If a fixed width is set for the table or column, the CELLSPACING and CELLPADDING may demand more space than assigned. Current practice is for the latter to take precedence over WIDTH attributes when a conflict occurs, although this isn't required by this specification.

Table Captions

```
<!ELEMENT caption - - (%text;)+>

<!ENTITY % Caption "(top|bottom|left|right)">

<!ATTLIST caption                  -- table caption --
        %attrs;                    -- id, lang, dir and class --
        align   %Caption; #IMPLIED -- relative to table --
        >
```

The optional CAPTION element is used to provide a caption for the table. Both start and end tags are required.

ID, CLASS, LANG and DIR
: See earlier description of common attributes.

ALIGN
: This may be used to control the placement of captions relative to the table. When present, the ALIGN attribute should have one of the values: TOP, BOTTOM, LEFT and RIGHT. It is recommended that the caption is made to fit within the width or height of the table as appropriate. The default position of the caption is deliberately unspecified.

  Note the ALIGN attribute is overused in HTML, but is retained here for compatibility with currently deployed browsers.

The COLGROUP Element

```
<!ELEMENT colgroup - O (col*)>

<!ATTLIST colgroup
        %attrs;                    -- id, lang, dir and class --
        span    NUMBER   1         -- default number of columns in --
                                   -- group --
        width   CDATA    #IMPLIED  -- default width for enclosed --
                                   -- COLs --
        %cell.halign;              -- horizontal alignment in --
                                   -- cells --
```

```
        %cell.valign;              -- vertical alignment in cells --
        >
```

The COLGROUP element acts as a container for a group of columns, and allows you to set default properties for these columns. In the absence of a COLGROUP element, all columns in the table are assumed to belong to a single column group. Each COLGROUP element can contain zero or more COL elements. COLGROUP requires a start tag, but the end tag may be omitted. This is useful when defining a sequence of COLGROUP elements, e.g.

```
<TABLE FRAME=BOX RULES=COLS>
  <COLGROUP>
    <COL WIDTH="1*">
    <COL WIDTH="2*">
  <COLGROUP>
    <COL WIDTH="1*">
    <COL WIDTH="3*">
  <THEAD>
    <TR> ...
</TABLE>
```

COLGROUP elements can be used with the following attributes:

ID, CLASS, LANG and DIR
: See earlier description of common attributes.

SPAN
: A positive integer value that specifies a default for how many columns are in this group. This attribute should be ignored if the COLGROUP element contains one or more COL elements. It provides a convenient way of grouping columns without the need to supply COL elements.

WIDTH
: Specifies a default width for each of the grouped columns, see standard units. In addition, the "*" suffix denotes relative widths, e.g.

```
     width=64        width in screen pixels
     width=0.5*      a relative width of 0.5
```

: Relative widths act as constraints on the relative widths of different columns. If a COLGROUP element specifies a relative width of zero, all of the columns in the group should be set to their minimum widths, unless they are associated with a COL element with an overriding WIDTH attribute. When widths are

given in absolute units, the user agent can use these to constrain the width of the table. The "*" suffix is used to simplify importing tables from the CALS representation.

ALIGN, CHAR, CHAROFF and VALIGN
Specify values for horizontal and vertical alignment within table cells. See inheritance order of alignment properties.

The COL Element

```
<!ELEMENT col - O EMPTY>

<!ATTLIST col                       -- column groups and --
                                    -- properties --
        %attrs;                     -- id, lang, dir and class --
        span    NUMBER   1          -- number of columns spanned --
                                    -- by group --
        width   CDATA    #IMPLIED   -- column width specification --
        %cell.halign;               -- horizontal alignment in --
                                    -- cells --
        %cell.valign;               -- vertical alignment in cells --
        >
```

This optional element is used to specify column based defaults for table properties. It is an empty element, and as such has no content, and shouldn't be given an end tag. Several COL elements may be given in succession. COL attributes override those of the parent COLGROUP element.

ID, CLASS, LANG and DIR
See earlier description of common attributes.

SPAN
A positive integer value that specifies how many columns this element applies to, defaulting to one. In the absence of SPAN attributes the first COL element applies to the first column, the second COL element to the second column and so on. If the second COL element had SPAN=2, it would apply to the second and third column. The next COL element would then apply to the fourth column and so on. SPAN=0 has a special significance and implies that the COL element spans all columns from the current column up to and including the last column. Note that a COL SPAN does not define a group. It is merely a way to share attribute definitions.

WIDTH
Specifies the width of the columns, see standard units. If the element spans several columns then the WIDTH attribute specifies the width for each of the individual columns - not the width of the span. In addition, the "*" suffix denotes relative widths,

e.g.

```
width=64        width in screen pixels
width=0.5*      a relative width of 0.5
```

Relative widths act as constraints on the relative widths of different columns. If a COL element specifies a relative width of zero, the column should always be set to its minimum width. When widths are given in absolute units, the user agent can use these to constrain the width of the table. The "*" suffix is used to simplify importing tables from the CALS representation.

ALIGN, CHAR, CHAROFF and VALIGN
Specify values for horizontal and vertical alignment within table cells. See inheritance order of alignment properties.

Table Head, Foot and Body Elements

```
<!ELEMENT thead - O tr+>
<!ELEMENT tfoot - O tr+>
<!ELEMENT tbody O O tr+>

<!ATTLIST (thead|tbody|tfoot)      -- table section --
        %attrs;                    -- id, lang, dir and class --
        %cell.halign;              -- horizontal alignment in --
                                   -- cells --
        %cell.valign;              -- vertical alignment in cells --
        >
```

Tables may be divided up into head and body sections. The THEAD and TFOOT elements are optional, but one or more TBODY elements are always required. If the table only consists of a TBODY section, the TBODY start and end tags may be omitted, as the parser can infer them. If a THEAD element is present, the THEAD start tag is required, but the end tag can be omitted, provided a TFOOT or TBODY start tag follows. The same applies to TFOOT.

Note: This definition provides compatibility with tables created for the older model, as well as allowing the end tags for THEAD, TFOOT and TBODY to be omitted.

The THEAD, TFOOT and TBODY elements provide a convenient means for controlling rendering. If the table has a large number of rows in the body, user agents may choose to use a scrolling region for the table body sections. When rendering to a paged device, tables will often have to be broken across page boundaries. The THEAD, TFOOT and TBODY elements allow the user agent to repeat the table foot at the bottom of the current page, and then the table head at the top of the new page before continuing on with the table body.

TFOOT is placed before the TBODY in the markup sequence, so that browsers can render the foot before receiving all of the table data. This is useful when very long tables are rendered with scrolling body sections, or for paged output, involving breaking the table over many pages.

Each THEAD, TFOOT and TBODY element must contain one or more TR elements.

ID, CLASS, LANG and DIR
: See earlier description of common attributes.

ALIGN, CHAR, CHAROFF and VALIGN
: Specify values for horizontal and vertical alignment within table cells. See inheritance order of alignment properties.

Table Row (TR) elements

```
<!ELEMENT tr - O (th|td)+>

<!ATTLIST tr                        -- table row --
        %attrs;                     -- id, lang, dir and class --
        %cell.halign;               -- horizontal alignment in --
                                    -- cells --
        %cell.valign;               -- vertical alignment in cells --
        >
```

The TR or table row element acts as a container for a row of table cells. The end tag may be omitted.

ID, CLASS, LANG and DIR
: See earlier description of common attributes.

ALIGN, CHAR, CHAROFF and VALIGN
: Specify values for horizontal and vertical alignment within table cells. See inheritance order of alignment properties.

Table Cells: TH and TD

```
<!ELEMENT (th|td) - O %body.content>

<!ATTLIST (th|td)                   -- header or data cell --
        %attrs;                     -- id, lang, dir and class --
        axis    CDATA    #IMPLIED   -- defaults to cell content --
        axes    CDATA    #IMPLIED   -- list of axis names --
        nowrap (nowrap)  #IMPLIED   -- suppress word wrap --
        rowspan NUMBER   1          -- number of rows spanned by --
                                    -- cell --
        colspan NUMBER   1          -- number of cols spanned by --
                                    -- cell --
        %cell.halign;               -- horizontal alignment in --
                                    -- cells --
        %cell.valign;               -- vertical alignment in cells --
        >
```

TH elements are used to represent header cells, while TD elements are used to represent data cells. This allows user agents to render header and data cells distinctly, even in the absence of style sheets.

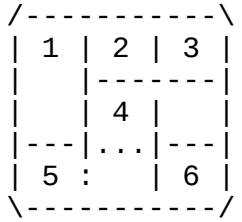
Cells can span multiple rows and columns, and may be empty. Cells spanning rows contribute to the column count on each of the spanned rows, but only appear in the markup once (in the first row spanned). The row count is determined by the number of TR elements. Any rows implied by cells spanning rows beyond this should be ignored.

If the column count for the table is greater than the number of cells for a given row (after including cells for spanned rows), the missing cells are treated as occurring on the right hand side of the table and rendered as empty cells. If the language context indicates a right to left writing order, then the missing cells should be placed on the left hand side.

It is possible to create tables with overlapping cells, for instance:

```
<table border>
<tr><td rowspan=2>1<td>2<td>3
<tr><td rowspan=2>4
<tr><td colspan=2>5<td>6
</table>
```

which might look something like:



In this example, the cells labelled 4 and 5 overlap. In such cases, the rendering is implementation dependent.

The AXIS and AXES attributes for cells provide a means for defining concise labels for cells. When rendering to speech, these attributes may be used to provide abbreviated names for the headers relevant to each cell. Another application is when you want to be able to later process table contents to enter them into a database. These attributes are then used to give database field names. The table's class attribute should be used to let the software recognize which tables can be treated in this way.

ID, CLASS, LANG and DIR
: See earlier description of common attributes.

AXIS
: This defines an abbreviated name for a header cell, e.g. which can be used when rendering to speech. It defaults to the cell's content.

AXES
: This is a comma separated list of axis names which together identify the row and column headers that pertain to this cell. It is used for example when rendering to speech to identify the cell's position in the table. If missing the user agent can try to follow up columns and left along rows (right for some languages) to find the corresponding header cells.

NOWRAP, e.g. <TD NOWRAP>
: The presence of this attribute disables automatic wrapping of text lines for this cell. If used uncautiously, it may result in excessively wide cells. This attribute is defined for backwards compatibility with deployed user agents. Greater control is possible with associated style sheet languages (for example for control over overflow handling).

ROWSPAN, e.g. <TD ROWSPAN=2>
A positive integer value that defines how may rows this cell spans. The default ROWSPAN is 1. ROWSPAN=0 has a special significance and implies that the cell spans all rows from the current row up to the last row of the table.

COLSPAN, e.g. <TD COLSPAN=2>
A positive integer value that defines how may columns this cell spans. The default COLSPAN is 1. COLSPAN=0 has a special significance and implies that the cell spans all columns from the current column up to the last column of the table.

ALIGN, CHAR, CHAROFF and VALIGN
Specify values for horizontal and vertical alignment within table cells. See inheritance order of alignment properties.

Note: It is recommended that implementors provide support for the Netscape 1.1 WIDTH attribute for TH and TD, although this isn't part of the current specification. Document authors are advised to use the width attribute for the COL element instead.

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.

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. As line wrap has been disabled, paragraphs are treated as long lines unless broken by
 elements. The minimum width is given by the widest word or image etc. taking into account leading indents and list bullets etc. In other words, if you were to format the cell’s content in a window of its own, determine the minimum width you could make the window 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 of 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 which span multiple columns, a simple approach, as used by Arena, is to evenly apportion the min/max widths 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. Experimental study suggests a blend of the two approaches will give 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. enclosing) 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.

HTML Table DTD

The DTD or document type definition provides the formal definition of the allowed syntax for HTML tables.

```
<!-- Content model entities imported from parent DTD:

  %body.content; allows table cells to contain headers, paras,
  lists, form elements and even arbitrarily nested tables.

  %text; is text characters, including character entities and
  character emphasis elements, IMG and anchors
-->

<!ENTITY % attrs
       "id      ID       #IMPLIED  -- element identifier --
        class   NAMES    #IMPLIED  -- for subclassing elements --
        lang    NAME     #IMPLIED  -- as per RFC 1766 --
        dir   (ltr|rtl)  #IMPLIED  -- I18N text direction --">

<!--
 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, its appropriate to treat this as frame=border for backwards
 compatibility with deployed browsers.
-->

<!ENTITY % Frame "(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 % Rules "(none | groups | rows | cols | all)">

<!-- horizontal placement of table relative to window -->
<!ENTITY % Where "(left|center|right)">

<!-- horizontal alignment attributes for cell contents -->
<!ENTITY % cell.halign
        "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 % cell.valign
        "valign  (top|middle|bottom|baseline)  #IMPLIED"
        >

<!ELEMENT table - - (caption?, (col*|colgroup*), thead?, tfoot?, t
                     body+)>
<!ELEMENT caption - - (%text;)+>
<!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 %body.content>

<!ATTLIST table                    -- table element --
        %attrs;                    -- id, lang, dir and class --
        align   %Where;  #IMPLIED  -- table position relative to --
                                   -- window --
        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   %Frame;  #IMPLIED  -- which parts of table frame to --
                                   -- include --
        rules   %Rules;  #IMPLIED  -- rulings between rows and cols --
        cellspacing CDATA #IMPLIED -- spacing between cells --
        cellpadding CDATA #IMPLIED -- spacing within cells --
```

```
        >

<!-- ALIGN is used here for compatibility with deployed browsers -->
<!ENTITY % Caption "(top|bottom|left|right)">

<!ATTLIST caption                  -- table caption --
        %attrs;                    -- id, lang, dir and class --
        align  %Caption; #IMPLIED  -- relative to table --
        >

<!--
COLGROUP groups a set of COL elements. It allows you to group
several columns together.
-->
<!ATTLIST colgroup
        %attrs;                    -- id, lang, dir and class --
        span    NUMBER   1         -- default number of columns in --
                                   -- group --
        width   CDATA    #IMPLIED  -- default width for enclosed COLs --
        %cell.halign;              -- horizontal alignment in cells --
        %cell.valign;              -- 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;                    -- id, lang, dir and class --
        span    NUMBER   1         -- number of columns spanned by --
                                   -- group --
        width   CDATA    #IMPLIED  -- column width specification --
        %cell.halign;              -- horizontal alignment in cells --
        %cell.valign;              -- vertical alignment in cells --
        >

<!--
    Use THEAD to duplicate headers when breaking table
    across page boundaries, or for static headers when
    body sections are rendered in scrolling panel.

    Use TFOOT to duplicate footers when breaking table
    across page boundaries, or for static footers when
```

```
    body 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;                    -- id, lang, dir and class --
        %cell.halign;              -- horizontal alignment in cells --
        %cell.valign;              -- vertical alignment in cells --
        >

<!ATTLIST tr                       -- table row --
        %attrs;                    -- id, lang, dir and class --
        %cell.halign;              -- horizontal alignment in cells --
        %cell.valign;              -- vertical alignment in cells --
        >

<!ATTLIST (th|td)                  -- header or data cell --
        %attrs;                    -- id, lang, dir and class --
        axis    CDATA    #IMPLIED  -- defaults to cell content --
        axes    CDATA    #IMPLIED  -- list of axis names --
        nowrap (nowrap)  #IMPLIED  -- suppress word wrap --
        rowspan NUMBER   1         -- number of rows spanned by cell --
        colspan NUMBER   1         -- number of cols spanned by cell --
        %cell.halign;              -- horizontal alignment in cells --
        %cell.valign;              -- vertical alignment in cells --
        >
```

References

Arena
: W3C's HTML3 browser, see http://www.w3.org/pub/WWW/Arena/. Arena was originally created as a proof of concept demo for ideas in the HTML+ specification that preceded HTML3. The browser is now being re-implemented to provide a reference implementation of HTML3 along with support for style sheets and client-side scripting.

CALS
: Continuous Acquisition and Life-Cycle Support (formerly Computer-aided Acquisition and Logistics Support) (CALS) is a Department of Defense (DoD) strategy for achieving effective creation, exchange, and use of digital data for weapon systems and equipment. More information can be found from the US Navy CALS home page at http://navysgml.dt.navy.mil/cals.html

HTML 2.0 (RFC1866)
Hypertext Markup Language Specification Version 2.0 by T. Berners-Lee and D. Connolly, November 1995. Further information can be found at http://www.w3.org/pub/WWW/MarkUp/ or at ftp://ds.internic.net/rfc/rfc1866.txt

HTML 3.0
Hypertext Markup Language Specification Version 3.0. The initial draft specification as published in March 1995. Work on refining HTML3 is proceeding piecemeal with the new table specification as one of the pieces. For W3C related work on HTML, see http://www.w3.org/pub/WWW/MarkUp/.

RFC 1766
"Tags for the Identification of Languages", by H. Alvestrand, UNINETT, March 1995. This document can be downloaded from ftp://ds.internic.net/rfc/rfc1766.txt.

Security Considerations

Security issues are not discussed in this memo.

Author's Address

Dave Raggett W3C

EMail: dsr@w3.org

The World Wide Web Consortium: http://www.w3.org/