Freebox HTML Reference Specification

Revision: 1.4

February 16, 2005

Status of this document

This document has a reference HTML-specification of freebox user agent. It has a reference to freebox user agent (firmware branche 1.6). It is derived from *W3C HTML-3.2 reference specification*[1].

Abstract

The main application of the freeboxTV software is a customize HTML user agent. The HTML (HyperText Markup Language) is a simple markup language used to create hypertext documents. It has generic semantics that are appropriate for representing information from a wide range of applications.

[1] HTML 3.2 Reference Specification, http://www.w3.org/TR/REC-html32

Contents

1 Introduction

1.1 HTML-3.2

HTML is W3C's specification for HTML, developed in early '96 together with vendors including IBM, Microsoft, Netscape Communications Corporation, Novell, SoftQuad, Spyglass, and Sun Microsystems. HTML 3.2 adds widely deployed features such as tables, applets and text flow around images, while providing full backwards compatibility with the existing standard HTML 2.0[2].

1.2 HTML as an SGML Application

HTML 3.2 is an SGML application conforming to International Standard ISO 8879 – Standard Generalized Markup Language. As an SGML application, the syntax of conforming HTML 3.2 documents is defined by the combination of the *SGML declaration* and the *document type definition* (DTD). This specification defines the intended interpretation of HTML 3.2 elements, and places further constraints on the permitted syntax which are otherwise inexpressible in the DTD.

The SGML rules for record boundaries are tricky. In particular, a record end immediately following a start tag should be discarded. For example:

```
<P>
Text
```

is equivalent to:

```
<P>Text
```

Similarly, a record end immediately preceding an end tag should be discarded. For example:

```
Text
</P>
```

is equivalent to:

```
Text</P>
```

Except within literal text (e.g. the `PRE` element), HTML treats contiguous sequences of white space characters as being equivalent to a single space character (ASCII decimal 32). These rules allow authors considerable flexibility when editing the marked-up text directly. Note that future revisions to HTML may allow for the interpretation of the horizontal tab character (ASCII decimal 9) with respect to a tab rule defined by an associated style sheet.

SGML entities in PCDATA content or in CDATA attributes are expanded by the parser, e.g. `&#233;` is expanded to the ISO Latin-1 character decimal 233 (a lower case letter e with an acute accent). This could also have been written as a named character entity, e.g. `&eacute;`. The `&` character can be included in its own right using the named character entity `&`.

HTML allows CDATA attributes to be unquoted provided the attribute value contains only letters (a to z and A to Z), digits (0 to 9), hyphens (ASCII decimal 45) or, periods (ASCII decimal 46). Attribute values can be quoted using double or single quote marks (ASCII decimal 34 and 39 respectively). Single quote marks can be included within the attribute value when the value is delimited by double quote marks, and vice versa.

[2]RFC 1866, *Hypertext Markup Language - 2.0.* T. Berners-Lee, D. Connolly.

1.3 The Structure of HTML documents

HTML Documents start with a `DOCTYPE` declaration followed by an HTML element containing a `HEAD` and then a `BODY` element:

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 3.2 Final//EN">
  <HTML>
  <HEAD>
  <TITLE>A study of population dynamics</TITLE>
  ... other head elements
  </HEAD>
  <BODY>
  ... document body
  </BODY>
  </HTML>
```

In practice, the `HTML`, `HEAD` and `BODY` start and end tags can be omitted from the markup as these can be inferred in all cases by parsers conforming to the HTML 3.2 DTD.

2 The HEAD element

This contains the document head, but you can always omit both the start and end tags for `HEAD`. The contents of the document head is an unordered collection of the following elements:

- The TITLE element
- The STYLE element
- The SCRIPT element
- The BASE element
- The META element
- The LINK element
- The FLAGS element

```
<!ENTITY % head.content "TITLE & BASE?">
<!ENTITY % head.misc "SCRIPT|STYLE|META|LINK|SERVICES|FLAGS">

<!ELEMENT HEAD O O  (%head.content) +(%head.misc)>
```

The `%head.misc` entity is used to allow the associated elements to occur multiple times at arbitrary positions within the HEAD. The following elements can be part of the document head:

- TITLE defines the document title, and is always needed.
- BASE defines base URL for resolving relative URLs.
- SCRIPT reserved for future use with scripting languages.
- STYLE reserved for future use with style sheets.
- META used to supply meta info as name/value pairs.
- LINK used to define relationships with other documents.
- FLAGS used to define specifics flags of this page.

`TITLE`, `SCRIPT STYLE` and `SERVICES` are containers and require both start and end tags. The other elements are not containers so that end tags are forbidden.

2.1 TITLE

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

Every HTML document **must** have exactly one `TITLE` element in the document's `HEAD`. The content model is PCDATA. As a result, character entities can be used for accented characters and to escape special characters such as `&` and `<`. Markup is not permitted in the content of a `TITLE` element.

Example TITLE element:

```
<TITLE>Channel 4</TITLE>
```

2.2 STYLE and SCRIPT

```
<!ELEMENT STYLE  - - CDATA -- placeholder for style info -->
<!ELEMENT SCRIPT - - CDATA -- placeholder for script statements -->
```

These are place holders for the introduction of style sheets and client-side scripts in future versions of HTML. User agent hide the contents of these elements.

These elements are defined with CDATA as the content type. As a result they may contain only SGML characters. All markup characters or delimiters are ignored and passed as data to the application, except for ETAGO ("</") delimiters followed immediately by a name character [a-zA-Z]. This means that the element's end-tag (or that of an element in which it is nested) is recognized, while an error occurs if the ETAGO is invalid.

2.3 BASE

```
<!ELEMENT BASE - O EMPTY>
<!ATTLIST BASE
    href %URL  #REQUIRED
    >
```

The `BASE` element gives the base URL for dereferencing relative URLs, using the rules given by the URL specification, e.g.

```
<BASE href="http://192.168.2.10/channel/france5/band.html">
     ...
<IMG SRC="icons/logo.gif">
```

The image is deferenced to

```
http://192.168.2.10/channel/france5/icons/logo.gif
```

In the absence of a `BASE` element the document URL should be used. Note that this is not necessarily the same as the URL used to request the document, as the base URL may be overridden by an HTTP header accompanying the document.

2.4 META

```
<!ELEMENT META - O EMPTY    -- Generic Metainformation -->
<!ATTLIST META
    name        NAME    #IMPLIED  -- metainformation name --
    content     CDATA   #REQUIRED -- associated information --
    >
```

The `META` element can be used to include name/value pairs describing properties of the document, such as author, expiry date, a list of key words etc. The `NAME` attribute specifies the property name while the `CONTENT` attribute specifies the property value, e.g.

```
<META NAME="refresh" CONTENT="4;url=timeout.html">
```

Here are some proposed properties values:

- **disappear**: Defines a timeout before to disappear the current page. Timeout start when the first video picture is displayed.
- **refresh**: Defines a timeout before refreash this page, or load a new one. To define a new page, add `url=` declaration in content, e.g.

```
<meta name="refresh" content="4;url=refresh.html">
```

- **nopicture**: Refresh or load a new page if no image is decoded during a specified seconds. To define a new page, add `url=` declaration in content, e.g.

```
<meta name="nopicture" content="4;url=nopicture.html">
```

- **channel**: Specifies the current number of channel.
- **settings_page**: Defines URL of settings page (page loaded on freebox wake up).
- **channel_page**: Defines generic URL of channel page.
- **home_page**: Defines URL of page invoked when end user presses on key "home".
- **love_page**: Defines URL of page invoked when end user presses on key "love".
- **mail_page**: Defines URL of page invoked when end user presses on key "mail".
- **love2_page**: Defines URL of page invoked when end user longly presses on key "love".
- **mail2_page**: Defines URL of page invoked when end user longly presses on key "mail".

2.5 LINK

`LINK` provides a method to associate a page to key press action.

```
<!ELEMENT LINK - O EMPTY>
<!ATTLIST LINK
    href    %URL    #IMPLIED    -- URL for linked resource --
    rel     CDATA   #IMPLIED    -- forward link types --
    title   CDATA   #IMPLIED    -- advisory title string --
    >
```

- **href**
  Specifies a URL designating the linked resource.
- **rel**
  The forward relationship also known as the "link type". It specifies a named relationship from the enclosing document to the resource specified by the `HREF` attribute.
- **title**
  An advisory title for the linked resource.

The freebox user agent relationship values are:

- `rel=guide`: page associated with the key "guide"
- `rel=options`: page associated with the key "options"
- `rel=help`: page associated with the key "help"
- `rel=info`: page associated with the key "info"
- `rel=red`: page associated with the key "red"
- `rel=green`: page associated with the key "green"
- `rel=yellow`: page associated with the key "yellow"
- `rel=blue`: page associated with the key "blue"

- `rel=play`: page associated with the key “play”
- `rel=pause`: page associated with the key “pause”
- `rel=up`: page associated with the key “up”
- `rel=down`: page associated with the key “down”
- `rel=right`: page associated with the key “right”
- `rel=left`: page associated with the key “left”

Example `LINK` elements:

```
<link rel=guide href="/guide/index.html">
<link rel=options href="/options/index.html">
<LINK rel=info href="info.html">
```

2.6 FLAGS

```
<!ENTITY % Boolean "(true|false)">
<!ELEMENT FLAGS - O EMPTY>
<!ATTLIST FLAGS
    program_key        %Boolean  #IMPLIED    -- enable program keys --
    volume_key         %Boolean  #IMPLIED    -- enable volume keys --
    >
```

The `FLAGS` element is used to define specifics flags of this page. The flags are:

- **program_key**
  Enable/Disable program control keys.
- **volume_key**
  Enable/Disable volume control keys.

3 The BODY element

This contains the document body. Both start and end tags for `BODY` may be omitted. The body can contain a wide range of elements:

- Headings (H1 - H6)
- Block level Elements
- Text level elements

The key attributes are: `BACKGROUND`, `BGCOLOR`, `TEXT`, `LINK`, `VLINK` and `ALINK`. These can be used to set a background video program, plus background and foreground colors for normal text and hypertext links.

3.1 Body element

```
<!ENTITY % body.content "(%heading | %text | %block )*">
<!ENTITY % color "CDATA" -- a color specification -->
<!ENTITY % body-color-attrs "
    bgcolor %color  #IMPLIED
    text    %color  #IMPLIED
    link    %color  #IMPLIED
    vlink   %color  #IMPLIED
    alink   %color  #IMPLIED
    family  CDATA   #IMPLIED
    ">
<!ELEMENT BODY O O  %body.content>
<!ATTLIST BODY
    background %URL #IMPLIED  -- video stream tile for document background --
    %body-color-attrs;  -- bgcolor, text, link, vlink, alink --
    >
```

Example:

```
<body text="#ffffff3F" link="#f0f0f03f" alink="#0000003f" vlink="#cc000030">
```

- **bgcolor**
  Specifies the background color for the document body. See below for the syntax of color values.
- **text**
  Specifies the color used to stroke the document's text. This is generally used when you have changed the background color with the `BGCOLOR` or `BACKGROUND` attributes.
- **link**
  Specifies the color used to stroke the text for unfocused hypertext links.
- **vlink**
  Specifies the color used to stroke the text for selected hypertext links.
- **alink**
  Specifies the highlight color used to stroke the text for hypertext links at the moment the user focuses on the link.
- **family**
  Specifies the default font family that will be used to tile the document.
- **background**
  Specifies a URL for background audio/video stream that will be used to tile the document background.

3.1.1 Colors

Colors are given in the sRGB color space and mixing alpha blended with video. Red, green and blue components are 8 bits hexadecimal numbers. Alpha blended mix weight is a 6 bits number allowing mixing ration from 0 to 1 with a resolution of 1/64. The resulting pixel can be completely transparent (weighting of (0/3F) or can completely cover the video (3F/3F).

Example:

```
COLOR="#C0FFC02A"
```

3.1.2 Background audio/video stream

A audio/video stream URL is defined by 3 components: protocol, address and arguments, example:

3.1.3 Children elements

Most elements that can appear in the document body fall into one of two groups: block level elements which cause paragraph breaks, and text level elements which don't. Common block level elements include `H1` to `H6` (headers), `P` (paragraphs) `LI` (list items), and `HR` (horizontal rules). Common text level elements include `EM`, `I`, `B` and `FONT` (character emphasis), `A` (hypertext links), `IMG` and `BR` (line breaks). Note that block elements generally act as containers for text level and other block level elements (excluding headings elements), while text level elements can only contain other text level elements. The exact model depends on the element.

3.2 Headings

```
<!ELEMENT ( %heading )  - -  (%text;)*>
<!ATTLIST ( %heading )
    align  (left|center|right) #IMPLIED
    >
```

`H1`, `H2`, `H3`, `H4`, `H5` and `H6` are used for document headings. You always need the start and end tags. `H1` elements are more important than `H2` elements and so on, so that `H6` elements define the least important level of headings. More important headings are generally rendered in a larger font than less important ones. Use the optional `ALIGN` attribute to set the text alignment within a heading, e.g.

```
<H1 ALIGN=CENTER> <i>... centered heading ...</i> </H1>
```

The default is left alignment, but this can be overridden by an enclosing `DIV` or `CENTER` element.

3.3 Block elements

- **P** *paragraphs*
  The paragraph element requires a start tag, but the end tag can always be omitted. Use the `ALIGN` attribute to set the text alignment within a paragraph, e.g. `<P ALIGN=RIGHT>`
- **UL** *unordered lists*
  These require start and end tags, and contain one or more `LI` elements representing individual list items.
- **DL** *definition lists*
  These require start and end tags and contain `DT` elements that give the terms, and `DD` elements that give corresponding definitions.
- **PRE** *preformatted text*
  Requires start and end tags. These elements are rendered with a monospaced font and preserve layout defined by whitespace and line break characters.

- **DIV** *document divisions*
  Requires start and end tags. It is used with the `ALIGN` attribute to set the text alignment of the block elements it contains. `ALIGN` can be one of `LEFT`, `CENTER` or `RIGHT`.
- **CENTER** *text alignment*
  Requires start and end tags. It is used to center text lines enclosed by the `CENTER` element. See `DIV` for a more general solution.
- **FORM** *fill-out forms*
  Requires start and end tags. This element is used to define a fill-out form for processing by HTTP servers. The attributes are `ACTION`, `METHOD` .
- **HR** *horizontal rules*
  Not a container, so the end tag is forbidden. attributes are `ALIGN`, `NOSHADE`, `SIZE` and `WIDTH`.
- **TABLE** *can be nested*
  Requires start and end tags. Each table has one or more `TR` elements defining table rows. Each row has one or more cells defined by `TH` or `TD` elements. Attributes for `TABLE` elements are `WIDTH`, `BORDER`, `CELLSPACING` and `CELLPADDING`.

3.3.1 Paragraphs

```
<!ELEMENT P     - O (%text)*>
<!ATTLIST P
    align  (left|center|right) #IMPLIED
    >
```

The `P` element is used to markup paragraphs. It is a container and requires a start tag. The end tag is optional as it can always be inferred by the parser. User agent place paragraph breaks before and after `P` elements.

Example:

```
<P>This is the first paragraph.
<P>This is the second paragraph.
```

Paragraphs are usually rendered flush left with a ragged right margin. The `ALIGN` attribute can be used to explicitly specify the horizontal alignment:

- `align=left` The paragraph is rendered flush left.
- `align=center` The paragraph is centered.
- `align=right` The paragraph is rendered flush right.

For example:

```
<p align=center>This is a centered paragraph.
<p align=right>and this is a flush right paragraph.
```

The default is left alignment, but this can be overridden by an enclosing `DIV` or `CENTER` element.

3.3.2 Unordered Lists

```
<!ELEMENT UL - -  (LI)+>
<!ELEMENT LI - O %flow -- list item -->
```

Unordered lists take the form:

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

The UL element is used for unordered lists. Both start and end tags are always needed. The LI element is used for individual list items. The end tag for LI elements can always be omitted. Note that LI elements can contain nested lists.

3.3.3 Definition Lists

```
<!ELEMENT DL    - -  (DT|DD)+>
<!ELEMENT DT - O  (%text)*>
<!ELEMENT DD - O  %flow;>
```

Definition lists take the form:

```
<DL>
  <DT>term name
    <DD>term definition
    ...
</DL>
```

DT elements can only act as containers for text level elements, while DD elements can hold block level elements as well, excluding headings elements.

3.3.4 Preformatted Text

```
<!ELEMENT PRE - - (%text)* -(%pre.exclusion)>
<!ATTLIST PRE>
```

The PRE element can be used to include preformatted text. User agent render this in a fixed pitch font, preserving spacing associated with white space characters such as space and newline characters. Automatic word-wrap should be disabled within PRE elements.

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

3.3.5 DIV and CENTER

```
<!ELEMENT DIV - - %body.content>
<!ATTLIST DIV
    align   (left|center|right) #IMPLIED -- alignment of following text --
    >
<!ELEMENT center - - %body.content>
```

DIV elements can be used to structure HTML documents as a hierarchy of divisions. The ALIGN attribute can be used to set the default horizontal alignment for elements within the content of the DIV element. Its value is restricted to LEFT, CENTER or RIGHT, and is defined in the same way as for the paragraph element <P>.

Note that because DIV is a block-like element it will terminate an open P element. CENTER is directly equivalent to DIV with ALIGN=CENTER. Both DIV and CENTER require start and end tags.

3.3.6 FORM

```
<!ENTITY % HTTP-Method "GET | POST" -- as per HTTP specification -->
<!ELEMENT FORM - - %body.content -(FORM)>
<!ATTLIST FORM
    action %URL #IMPLIED  -- server-side form handler --
    method (%HTTP-Method) GET -- see HTTP specification --
    >
```

This is used to define an HTML form, and you can have more than one form in the same document. Both the start and end tags are required. Forms can contain a wide range of HTML markup including several kinds of *form fields* such as single text fields, radio button groups, and checkboxes.

- **action** This specifies a URL which is either used to invoke a server-side forms handler via HTTP, e.g. `action="http://192.168.2.10/scripts/register.pl"`
- **method** When the action attribute specifies an HTTP server, the method attribute determines which HTTP method will be used to send the form's contents to the server. It can be either `GET` or `POST`, and defaults to `GET`. `POST` is not yet implemented.

3.3.7 HR - *horizontal rules*

Horizontal rules may be used to indicate a change in topic. In a speech based user agent, the rule could be rendered as a pause.

```
<!ELEMENT HR    - O EMPTY>
<!ATTLIST HR
    size  %Pixels #IMPLIED
    width %Length #IMPLIED
    >
```

`HR` elements are not containers so the end tag is forbidden. The attributes are:

- **size** This can be used to set the height of the rule in pixels.
- **width** This can be used to set the width of the rule in pixels (e.g. `width=100`) or as the percentage between the current left and right margins (e.g. `width="50%"`). The default is 100%.

3.3.8 Tables

HTML 3.2 includes a widely deployed subset of the specification given in RFC 1942[3] and can be used to markup tabular material or for layout purposes.

```
<!ENTITY % Where "(left|center|right)">
<!ENTITY % cell.halign "(left|center|right)">
<!ENTITY % cell.valign "(top|middle|bottom)">
<!ELEMENT table - - (tr)+>
<!ELEMENT tr - O (th|td)*>
<!ELEMENT (th|td) - O %body.content>
<!ATTLIST table                     -- table element --
    align       %Where;   #IMPLIED  -- table position relative to window --
    width       %Length   #IMPLIED  -- table width relative to window --
    border      %Pixels   #IMPLIED  -- controls frame width around table --
```

[3]RFC 1942, *HTML Tables.* D. Raggett. May 1996.

```
    cellspacing %Pixels   #IMPLIED  -- spacing between cells --
    cellpadding %Pixels   #IMPLIED  -- spacing within cells --
    bgcolor     %color    #IMPLIED  -- background color for cells --
    bordercolor %color    #IMPLIED  -- border color for cells --
    >
<!ATTLIST tr                    -- table row --
    align   %cell.halign; #IMPLIED  -- horizontal alignment in cells --
    valign  %cell.valign; #IMPLIED  -- vertical alignment in cells --
    bgcolor %color        #IMPLIED  -- background color for row --
    >
<!ATTLIST (th|td)                   -- header or data cell --
    rowspan NUMBER        1         -- number of rows spanned by cell --
    colspan NUMBER        1         -- number of cols spanned by cell --
    align   %cell.halign; #IMPLIED  -- horizontal alignment in cells --
    valign  %cell.valign; #IMPLIED  -- vertical alignment in cells --
    width   %Pixels       #IMPLIED  -- suggested width for cell --
    height  %Pixels       #IMPLIED  -- suggested height for cell --
    bgcolor %color        #IMPLIED  -- background color for cell --
    >
```

Tables take the general form:

```
<TABLE BORDER=3 CELLSPACING=2 CELLPADDING=2 WIDTH="80%">
<TR><TD> first cell <TD> second cell
<TR> ...
...
</TABLE>
```

The attributes on `TABLE` are all optional. By default, the table is rendered without a surrounding border. The table is generally sized automatically to fit the contents, but you can also set the table width using the `WIDTH` attribute. `BORDER`, `CELLSPACING` and `CELLPADDING` provide further control over the table's appearence.

Each table row is contained in a `TR` element, although the end tag can always be omitted. Table cells are defined by `TD` elements for data and `TH` elements for headers. Like `TR`, these are containers and can be given without trailing end tags. `TH` and `TD` support several attributes: `ALIGN` and `VALIGN` for aligning cell content, `ROWSPAN` and `COLSPAN` for cells which span more than one row or column. A cell can contain a wide variety of other block and text level elements including form fields and other tables.

The `TABLE` element always requires both start and end tags. It supports the following attributes:

- **align**
  This takes one of the case insensitive values: `LEFT`, `CENTER` or `RIGHT`. It specifies the horizontal placement of the table relative to the current left and right margins. It defaults to left alignment, but this can be overridden by an enclosing `DIV` or `CENTER` element.

- **width**
  In the absence of this attribute the table width is automatically determined from the table contents. You can use the `WIDTH` attribute to set the table width to a fixed value in pixels (e.g. `WIDTH=212`) or as a percentage of the space between the current left and right margins (e.g. `WIDTH="80%"`).

- **border**
  This attribute can be used to specify the width of the outer border around the table to a given number of pixels (e.g. `BORDER=4`). The value can be set to zero to suppress the border altogether. In the absence of this attribute the border should be suppressed.

- **cellspacing**
  In traditional desktop publishing software, adjacent table cells share a common border. This is not the case in HTML. Each cell is given its own border which is separated from the borders around neighboring cells. This separation can be set in pixels using the `CELLSPACING` attribute, (e.g. `CELLSPACING=10`). The same value also determines the separation between the table border and the borders of the outermost cells.

- **cellpadding**
  This sets the padding in pixels between the border around each cell and the cell's contents.

- **bgcolor**
  Specifies the background color for the table.

- **bordercolor**
  Specifies the background color for the border of the table.

The `TR` or table row element requires a start tag, but the end tag can always be left out. `TR` acts as a container for table cells. It has 3 attributes:

- **align**
  Sets the default horizontal alignment of cell contents. It takes one of the case insensitive values: `LEFT`, `CENTER` or `RIGHT` and plays the same role as the `ALIGN` attribute on paragraph elements.

- **valign**
  This can be used to set the default vertical alignment of cell contents within each cell. It takes one of the case insensitive values: `TOP`, `MIDDLE` or `BOTTOM` to position the cell contents at the top, middle or bottom of the cell respectively.

- **bgcolor**
  Specifies the background color for the row. It overrides the bgcolor of the table.

There are two elements for defining table cells. `TH` is used for header cells and `TD` for data cells. The start tags for `TH` and `TD` are always needed but the end tags can be left out. Table cells can have the following attributes:

- **rowspan**
  This takes a positive integer value specifying the number of rows spanned by this cell. It defaults to one.

- **colspan**
  This takes a positive integer value specifying the number of columns spanned by this cell. It defaults to one.

- **align**
  Specifies the default horizontal alignment of cell contents, and overrides the `ALIGN` attribute on the table row. It takes the same values: `LEFT`, `CENTER` and `RIGHT`. If you don't specify an `ALIGN` attribute value on the cell, the default is left alignment for `<td>` and center alignment for `<th>` although you can override this with an `ALIGN` attribute on the `TR` element.

- **valign**
  Specifies the default vertical alignment of cell contents, overriding the `VALIGN` attribute on the table row. It takes the same values: `TOP`, `MIDDLE` and `BOTTOM`. If you don't specify a VALIGN attribute value on the cell, the default is middle although you can override this with a `VALIGN` attribute on the `TR` element.

- **width**
  Specifies the suggested width for a cell content in pixels excluding the cell padding. This value will normally be used except when it conflicts with the width requirements for other cells in the same column.
- **height**
  Specifies the suggested height for a cell content in pixels excluding the cell padding. This value will normally be used except when it conflicts with the height requirements for other cells in the same row.
- **bgcolor**
  Specifies the background color for the cell. It overrides the bgcolor of the row.

Tables are commonly rendered in bas-relief, raised up with the outer border as a bevel, and individual cells inset into this raised surface. Borders around individual cells are only drawn if the cell has explicit content. White space doesn't count for this purpose with the exception of ` `.

The algorithms used to automatically size tables should take into account the minimum and maximum width requirements for each cell. This is used to determine the minimum and maximum width requirements for each column and hence for the table itself.

Cells spanning more than one column contribute to the widths of each of the columns spanned. One approach is to evenly apportion the cell's minimum and maximum width between these columns, another is to weight the apportioning according to the contributions from cells that don't span multiple columns.

The minimum and maximum width of nested tables contribute to the minimum and maximum width of the cell in which they occur. Once the width requirements are known for the top level table, the column widths for that table can be assigned. This allows the widths of nested tables to be assigned and hence in turn the column widths of such tables. If practical, all columns should be assigned at least their minimum widths. It is suggested that any surplus space is then shared out proportional to the difference between the minimum and maximum width requirements of each column.

3.4 Text level elements

These don't cause paragraph breaks. Text level elements that define character styles can generally be nested. They can contain other text level elements but not block level elements.

- Font style elements
- Phrase elements
- INPUT form Fields
- BOX element
- The A (anchor) element
- IMG - inline images
- FONT elements
- BR - line breaks

3.4.1 Font style elements

These all require start and end tags, e.g.

```
This has some <B>bold text</B>.
```

Text level elements must be properly nested - the following is in error:

```
This has some <B>bold and <I></B>italic text</I>.
```

User agent do their best to respect nested emphasis, e.g.

```
This has some <B>bold and <I>italic text</I></B>.
```

Where the available fonts are restricted or for speech output, alternative means should be used for rendering differences in emphasis.

- **I** italic text style
- **B** bold text style
- **U** underlined text style
- **BIG** places text in a large font
- **SMALL** places text in a small font

3.4.2 Phrase Elements

These all require start and end tags, e.g.

```
This has some <EM>emphasized text</EM>.
```

- **EM** basic emphasis typically rendered in an italic font
- **STRONG** strong emphasis typically rendered in a bold font

3.4.3 INPUT *text fields, radio buttons, check boxes*

`INPUT` can be used for a variety of form fields including single line text fields, password fields, checkboxes, radio buttons, submit buttons, hidden fields. `INPUT` elements are not containers and so the end tag is forbidden.

```
<!ENTITY % IAlign "(top|middle|bottom|left|right)">
<!ENTITY % InputType
    "(TEXT | PASSWORD | CHECKBOX | RADIO | SUBMIT | HIDDEN)">
<!ELEMENT INPUT - O EMPTY>
<!ATTLIST INPUT
    type %InputType TEXT     -- what kind of widget is needed --
    name  CDATA #IMPLIED     -- required for all but submit and reset --
    value CDATA #IMPLIED     -- required for radio and checkboxes --
    checked (checked) #IMPLIED -- for radio buttons and check boxes --
    size CDATA  #IMPLIED     -- specific to each type of field --
    maxlength NUMBER #IMPLIED
    align %IAlign #IMPLIED   -- vertical or horizontal alignment --
    >
```

- **type** Used to set the type of input field:
  - `type=text` *(the default)* A single line text field whose visible size can be set using the `size` attribute, e.g. `size=40` for a 40 character wide field. Users should be able to type more than this limit though with the text scrolling through the field to keep the input cursor in view. You can enforce an upper limit on the number of characters that can be entered with the `maxlength` attribute. The `name` attribute is used to name the field, while the `value` attribute can be used to initialize the text string shown in the field when the document is first loaded.

```
<input type=text size=40 name=user value="your name">
```

- `type=password` This is like `type=text`, but echoes characters using a character like * to hide the text from prying eyes when entering passwords. You can use `size` and `maxlength` attributes to control the visible and maximum length exactly as per regular text fields.

  ```
  <input type=password size=12 name=pw>
  ```

- `type=checkbox` Used for simple Boolean attributes, or for attributes that can take multiple values at the same time. The latter is represented by several checkbox fields with the same `name` and a different `value` attribute. Each checked checkbox generates a separate name/value pair in the submitted data, even if this results in duplicate names. Use the `checked` attribute to initialize the checkbox to its checked state.

  ```
  <input type=checkbox checked name=meddle value=yes>
  ```

- `type=radio` Used for attributes which can take a single value from a set of alternatives. Each radio button field in the group should be given the same `name`. Radio buttons require an explicit `value` attribute. Only the checked radio button in the group generates a name/value pair in the submitted data. One radio button in each group should be initially checked using the `checked` attribute.

  ```
  <input type=radio name=age value="0-12">
  <input type=radio name=age value="13-17">
  <input type=radio name=age value="18-25">
  <input type=radio name=age value="26-35" checked>
  <input type=radio name=age value="36-">
  ```

- `type=submit` This defines a button that users can click to submit the form's contents to the server. The button's label is set from the `value` attribute. If the `name` attribute is given then the submit button's name/value pair will be included in the submitted data. You can include several submit buttons in the form.

  ```
  <input type=submit value="Rock n Roll">
  ```

- `type=hidden` These fields should not be rendered and provide a means for servers to store state information with a form. This will be passed back to the server when the form is submitted, using the name/value pair defined by the corresponding attributes. This is a work around for the statelessness of HTTP.

  ```
  <input type=hidden name=customerid value="c2415-345-8563">
  ```

- **name** Used to define the property name that will be used to identify this field's content when it is submitted to the server.
- **value** Used to initialize the field, or to provide a textual label for submit.
- **checked** The presence of this attribute is used to initialize checkboxes and radio buttons to their checked state.
- **size** Used to set the visible size of text fields to a given number of average character widths, e.g. `size=20`
- **maxlength** Sets the maximum number of characters permitted in a text field.

3.4.4 BOX element

```
<!ENTITY % box.halign "(left|center|right)">
<!ENTITY % box.valign "(top|middle|bottom)">
<!ELEMENT BOX    - O EMPTY>
<!ATTLIST BOX
    bgcolor      %color      #IMPLIED
    abgcolor     %color      #IMPLIED
    bordercolor  %color      #IMPLIED
    textcolor    %color      #IMPLIED
    align        %Box.halign #IMPLIED
    valign       %Box.valign #IMPLIED
    width        %Pixels     #IMPLIED
    height       %Pixels     #IMPLIED
    border       %Pixels     #IMPLIED
    text         CDATA       #IMPLIED
    >
```

The box element is a the text level element. It can be used to define a button. Attributes are:

- **bgcolor**
  Specifies background color.
- **abgcolor**
  Specifies background color when box is focused.
- **bordercolor**
  Specifies border color. It overides the default text color.
- **textcolor**
  Specifies text color. It overides the default text color.
- **align**
  Specifies the horizontal alignment.
- **valign**
  Specifies the vertical alignment.
- **width**
  Specifies the width for the box in pixel.
- **height**
  Specifies the height for the box in pexel.
- **border**
  Specifies the width for the border of the box in pixel.
- **text**
  Specifies a textual label for the box.

3.4.5 The A (anchor) element

```
<!ELEMENT A - - (%text)* -(A)>
<!ATTLIST A
    href      %URL       #IMPLIED    -- URL for linked resource --
    title     CDATA      #IMPLIED    -- advisory title string --
    onfocus   %URL       #IMPLIED    -- URL invoke on focus --
    focused   (checked)  #IMPLIED    -- default focused link --
    tag       NUMBER     #IMPLIED
    >
```

Anchors can't be nested and always require start and end tags. They are used to define hypertext links and also to define named locations for use as targets for hypertext links, e.g.

```
<a href="/next_page.html">next page</a>
```

- **href**
  Specifies a URL acting as a network address for the linked resource. This could be another HTML document or an image etc.
- **title**
  An advisory title for the linked resource.
- **onfocus**
  Specifies a URL which is either used to invoke a server when link is focused.
- **focused**
  Force this link is focused when this page is displayed.
- **tag**
  Specifies a tag for this link.

3.4.6 IMG - *inline images*

```
<!ENTITY % IAlign "(top|middle|bottom|left|right)">
<!ELEMENT IMG    - O EMPTY --  Embedded image -->
<!ATTLIST IMG
    src     %URL     #REQUIRED  -- URL of image to embed --
    alt     CDATA    #IMPLIED   -- for display in place of image --
    height  %Pixels  #IMPLIED   -- suggested height in pixels --
    width   %Pixels  #IMPLIED   -- suggested width in pixels --
    >
```

Used to insert images. `IMG` is an empty element and so the end tag is forbidden. Images can be positioned vertically relative to the current textline .

```
<i>e.g.</i>  <IMG SRC="AC-DC.gif" ALT="Grand Canyon">
```

`IMG` elements support the following attributes:

- **src** This attribute is required for every `IMG` element. It specifies a URL for the image resource, for instance a GIF image file.
- **alt** This is used to provide a text description of the image .
- **width** Specifies the intended width of the image in pixels. When given together with the height, this allows user agent to reserve screen space for the image before the image data has arrived over the network.
- **height** Specifies the intended height of the image in pixels. When given together with the width, this allows user agent to reserve screen space for the image before the image data has arrived over the network.

3.4.7 FONT

```
<!ELEMENT FONT - - (%text)*     -- local change to font -->
<!ATTLIST FONT
    size    CDATA   #IMPLIED    -- [+]nn e.g. size="+1", size=4 --
    color   CDATA   #IMPLIED    -- #RRGGBBAA in hex --
    family  CDATA   #IMPLIED    -- font name --
    link    %color  #IMPLIED    -- redefine link color --
    alink   %color  #IMPLIED    -- redefine actived link color --
    vlink   %color  #IMPLIED    -- redefine visited link color --
    >
```

Requires start and end tags. This allows you to change the font size and/or color for the enclosed text. The attributes are: `SIZE` and `COLOR`. Font sizes are given in terms of a scalar range defined by the user agent with no direct mapping to point sizes etc.

- **size**
  This sets the font size for the contents of the font element. You can set size to an integer ranging from 1 to 6 for an absolute font size, or specify a relative font size with a signed integer value, e.g. `size="+1"` or `size="-2"`.
- **color**
  Used to set the color to stroke the text. Colors are given as RGB in hexadecimal notation .
- **family**
  Used to set a new font with the corresponding name.
- **link**
  It overrides `link` attribute of `BODY`.
- **alink**
  It overrides `alink` attribute of `BODY`.
- **vlink**
  It overrides `vlink` attribute of `BODY`.

3.4.8 BR

```
<!ELEMENT BR    - O EMPTY    -- forced line break -->
```

Used to force a line break. This is an empty element so the end tag is forbidden.