

Welcome to the HTML Reference Library

Please select a volume

- [1\) The HTML Language](#)
- [2\) Quick Reference Guide](#)
- [3\) Contacting the Author](#)
- [4\) New in this version](#)

This reference, using the Internet RFC as an [information base](#) is an on-line reference library of currently supported HTML elements - their syntax, and use.

It assumes that the user has knowledge of the World Wide Web and the various HTML user agents (browsers) available. Information on the broader topic of 'The World Wide Web' can be obtained by reading the [World Wide Web FAQ](#).

Stephen Le Hunte

RFC1866 - The RFC document specifies an Internet standards track protocol for the Internet community, and is a request for discussion and suggestions for improvements. The *official* internet draft HTML 3 specification has expired, with the HTML working group recently deciding to concentrate their efforts on separate, smaller sections of the specification.

Other information provided in this document has been taken from various sources, including the Netscape, NCSA Mosaic, and Microsoft World Wide Web sites, and the various HTML authoring Usenet newsgroups.

The World Wide Web FAQ is posted
(every four days) to the following UseNet
newsgroups:

- news.answers
- comp.infosystems.www.users
- comp.infosystems.www.providers
- comp.infosystems.www.misc
- comp.infosystems.gopher
- comp.infosystems.wais

The most recent version is also always held at :
http://sunsite.unc.edu/boutell/faq/www_faq.html

The FAQ is maintained by Thomas Boutell

HTML Language

The vast range of HTML MarkUp currently supported by available HTML user agents (Web browsers, such as Netscape, Mosaic etc.) can be divided into the following sections. Some elements featured here, may not be supported by all browsers. Where an element is known to be supported by specific browsers, the element description will be labelled as such.

[Document Structure Elements](#)

[Anchor Element](#)

[Block Formatting Elements](#)

[List Elements](#)

[Information type and Character formatting Elements.](#)

[Image Element](#)

[Form Elements](#)

[Table Elements](#)

[Character Data](#)

[Dynamic HTML Documents](#)

Recently, browsers (especially Netscape and the Internet Explorer) have implemented many extensions to HTML which are not featured in the expired draft specification of HTML 3. Care should be taken when using such extensions because they may only be supported by specific browsers. However, they offer significant advances to HTML authors and are likely to be used widely. The following elements represent such advances.

[Embedding Objects](#) - Embed Windows objects

[Client Side Image Maps](#) - Image maps processed by the client

[In Line Video](#) - Add in-line video clips

[HotJava](#) - Add audio or animation easily to HTML documents


[Frames](#) - Advanced page formatting for the Netscape Navigator

[Document soundtracks](#) - Internet Explorer background sounds.

[Highlighted scrolling text](#) - Internet Explorer Marquee scrolling text.

Also recently, a specification for [Server Side Includes](#) was released. While this is server specific, the 'tokens' that are employed in SSI, need to be included in the HTML document where the action is to be carried out, hence have been included in this document.



Wherever this symbol :  appears, a screen shot showing typical rendering of the element in question is available. To see the screenshot, click the picture.

Exactly...just like you did then.

Document Structure Elements

These elements are required within a HTML document. Apart from the [prologue document identifier](#), they represent the only HTML elements which are explicitly required for a document to conform to the standard.

The essential document structure elements are :

<HTML> ... </HTML>

<HEAD> ... </HEAD>

<BODY> ... </BODY>

In order to identify a document as HTML, each HTML document should start with the prologue:

```
<!DOCTYPE HTML PUBLIC "-//IETF//DTD HTML 2.0//EN">.
```

However, it is worth noting that if the document does not contain this type declaration, a HTML user agent should infer it. The above document identifier identifies the document as conforming to the HTML 2.0 DTD. There are separate [prologue identifiers](#) that should be used to define which particular DTD the document conforms to.

Prologue Identifiers

The following are various prologue identifiers that should be used in HTML documents. With the identifier is the name of the HTML DTD (document type definition) that the prologue identifier labels the HTML document as adhering to. I.e. a HTML document whose prologue identifier is `<!DOCTYPE HTML PUBLIC "-//IETF//DTD HTML Level 1//EN//">` should adhere to the `HTML-1.DTD` (see the RFC for HTML 2.0 (**RFC1866**) for the DTD). Such a document should not contain Form elements for example.

The document prologue identifier should be included before the `<HTML>` element of a HTML document, in fact, it should be the first line of any HTML document.

Ways to refer to Level 3: most general to most specific

```
<!DOCTYPE HTML PUBLIC "-//IETF//DTD HTML//EN//3.0">
<!DOCTYPE HTML PUBLIC "-//W30//DTD W3 HTML 3.0//EN//">
<!DOCTYPE HTML PUBLIC "-//IETF//DTD HTML 3.0//EN">
<!DOCTYPE HTML PUBLIC "-//IETF//DTD HTML 3.0//EN//">
<!DOCTYPE HTML PUBLIC "-//IETF//DTD HTML Level 3//EN">
<!DOCTYPE HTML PUBLIC "-//IETF//DTD HTML 3.0 Level 3//EN">
```

Ways to refer to strict Level 3: most general to most specific

```
<!DOCTYPE HTML PUBLIC "-//IETF//DTD HTML Strict//EN//3.0">
<!DOCTYPE HTML PUBLIC "-//W30//DTD W3 HTML Strict 3.0//EN//">
<!DOCTYPE HTML PUBLIC "-//IETF//DTD HTML Strict Level 3//EN">
<!DOCTYPE HTML PUBLIC "-//IETF//DTD HTML Strict 3.0 Level 3//EN">
```

NOTE : The above document identifiers shouldn't actually be used. They refer to a DTD in the draft HTML 3 specification which has expired. At present, no useable HTML 3 DTD actually exists. They have been included here for completeness.

Ways to refer to Level 2: most general to most specific

These all require conformance to the `HTML.DTD`.

```
<!DOCTYPE HTML PUBLIC "-//IETF//DTD HTML//EN">
<!DOCTYPE HTML PUBLIC "-//IETF//DTD HTML 2.0//EN">
<!DOCTYPE HTML PUBLIC "-//IETF//DTD HTML Level 2//EN">
<!DOCTYPE HTML PUBLIC "-//IETF//DTD HTML 2.0 Level 2//EN">
```

Ways to refer to Level 1: most general to most specific

These all require conformance to the `HTML-1.DTD`.

```
<!DOCTYPE HTML PUBLIC "-//IETF//DTD HTML Level 1//EN">
<!DOCTYPE HTML PUBLIC "-//IETF//DTD HTML 2.0 Level 1//EN">
```

Ways to refer to Strict Level 2: most general to most specific

These all require conformance to the `HTML-S.DTD`.

```
<!DOCTYPE HTML PUBLIC "-//IETF//DTD HTML Strict//EN">
<!DOCTYPE HTML PUBLIC "-//IETF//DTD HTML 2.0 Strict//EN">
<!DOCTYPE HTML PUBLIC "-//IETF//DTD HTML Strict Level 2//EN">
<!DOCTYPE HTML PUBLIC "-//IETF//DTD HTML 2.0 Strict Level 2//EN">
```


Ways to refer to Strict Level 1: most general to most specific
These all require conformance to the `HTML1-S.DTD`.

```
<!DOCTYPE HTML PUBLIC "-//IETF//DTD HTML Strict Level 1//EN">  
<!DOCTYPE HTML PUBLIC "-//IETF//DTD HTML 2.0 Strict Level 1//EN">
```

<HTML> ... </HTML>

This element identifies the document as containing HTML elements. It should immediately follow the [prologue document identifier](#) and serves to surround all of the remaining text, including all other elements. That is, the document should be constructed thus :

```
<!DOCTYPE HTML PUBLIC "-//IETF//DTD HTML 2.0//EN">
<HTML>
Here is all the rest of the document, including any elements.
</HTML>
```

The HTML element is not visible upon HTML user agent rendering and can contain only the [<HEAD>](#) and [<BODY>](#) elements.

<HEAD> ... </HEAD>

The head of an HTML document is an unordered collection of information about the document. It requires the Title element between <HEAD> and </HEAD> elements thus :

```
<HEAD>
<TITLE> Introduction to HTML </TITLE>
</HEAD>
```

The <HEAD> and </HEAD> elements do not directly affect the look of the document when rendered.

The following elements are related to the head element. While not directly affecting the look of the document when rendered, they do provide (if used) important information to the HTML user agent.

<BASE> - Allows base address of HTML document to be specified

<ISINDEX>- Allows keyword searching of the document

<LINK>- Indicate relationships between documents

<NEXTID>- Creates unique document identifiers

<TITLE>- Specifies the title of the document

<META> - Specifies document information useable by server/clients.

NOTE : The Title element is the **only** element described here that is **required** as part of the Head of a HTML document.

<BODY> ... </BODY>

[Attributes](#)

[See Also](#)

The body of a HTML document contains all the text and images that make up the page, together with all the HTML elements that provide the control/formatting of the page. The format is :

```
<BODY>
The document included here
</BODY>
```

The <BODY> and </BODY> elements do not directly affect the look of the document when rendered, although they are **required** in order for the document to conform to the specification standard.

First to be implemented by Netscape, the ability to specify background images and colours for HTML documents has recently been implemented by many other browsers. It should be noted that the following elements are not supported by every HTML user agent available.

Recent versions of the proposed HTML 3.0 spec. have added a **BACKGROUND** attribute to the BODY element. The purpose of this attribute is to specify a URL pointing to an image that is to be used as a background for the document. In many recent browsers, this background image is used to tile the full background of the document-viewing area. Thus specifying:

```
<BODY BACKGROUND="URL or path/filename.gif">
Document here
</BODY>
```

would cause whatever text, images, etc. appeared in that document to be placed on a background consisting of the (filename.gif) graphics file being tiled to cover the viewing area, much like bitmaps are used for Windows wallpaper.

The **BGCOLOR** attribute

This attribute to BODY is not currently in the proposed HTML 3.0 specification, but is supported by Netscape, the Internet Explorer, NCSA Mosaic and many other browsers and is being considered for inclusion in HTML 3.0. This element changes the colour of the background without having to specify a separate image that requires another network access to load. The format that is understood is:

```
<BODY BGCOLOR="#rrggbb">
Document here
</BODY>
```

Where "#rrggbb" is a hexadecimal red-green-blue triplet used to specify the background colour. See the [Colour Table](#) for examples of colours together with their #rrggbb values.

Microsoft's Internet Explorer also supports 16 default colours that can be referred to by name. For more information, see [](#).

Clearly, once the background colours/patterns have been changed, it will be necessary to also be able to control the foreground to establish the proper contrasts.

TEXT

This attribute is used to control the colour of all the normal text in the document. This basically consists of all text that is not specially coloured to indicate a link. The format of TEXT is the same as that of BGCOLOR.

```
<BODY TEXT="#rrggbb">
Document here
```

</BODY>

Microsoft's Internet Explorer also supports 16 default colours that can be referred to by name. For more information, see [](#)

LINK, VLINK, and ALINK attributes

These attributes let you control the colouring of link text. **VLINK** stands for visited link, and **ALINK** stands for active link. The default colouring of these is: **LINK**=blue, **VLINK**=purple, and **ALINK**=red. Again, the format for these attributes is the same as that for **BGCOLOR** and **TEXT**.

```
<BODY LINK="#rrggbb" VLINK="#rrggbb" ALINK="#rrggbb">
Document here
</BODY>
```

Microsoft's **Internet Explorer** also supports 16 default colours that can be referred to by name. For more information, see [](#)

The ability to *watermark* HTML documents, by fixing a background image so that it doesn't scroll as a normal background image does has been added to Microsoft's Internet Explorer since version 2.0. To give a page with a background image a watermark background, add **BGPROPERTIES=FIXED** to the **BODY** element as follows:

```
<BODY BACKGROUND="filename.gif" BGPROPERTIES=FIXED>
```

NOTE : This attribute is **Internet Explorer** specific.

LEFTMARGIN attribute

This **Internet Explorer** attribute allows the left hand margin of the document to be set. For example:

```
<BODY LEFTMARGIN="60">This document is indented 60 pixels from the left hand side
of the page</BODY>
```

TOPMARGIN attribute

This **Internet Explorer** specific attribute allows the top margin of the document to be set. For example:

```
<BODY TOPMARGIN="60">This document is indented 60 pixels from the top of the
page</BODY>
```

NOTE : Both of the above attributes can be set to 0, making the page start at the very top and very left hand side of the page.

Colouring Considerations.

Since these colour controls are all attributes of the **BODY** element, they can only be set once for the entire document. Document colour cannot be changed partially through a document.

Setting a background image requires the fetching of an image file from a second HTTP connection, it will **slow down** the perceived speed of document loading. **None** of the document can be displayed until the image is loaded and decoded. Needless to say, keep background images small.

If the Auto Load Images option is turned off, background images will not be loaded. If the background image is not loaded for any reason, and a **BGCOLOR** was not also specified, then any of the foreground controlling attributes (**LINK**, **VLINK**, and **ALINK**) will be ignored. The idea behind this is that if the requested background image is unavailable, or not loaded, setting requested text colours on top of the default grey background may make the document unreadable.

The following attributes are supported in the <BODY> element by the most recent browsers.

[BACKGROUND](#)

[TEXT](#)

[LINK](#)

[VLINK](#)

[ALINK](#)

[BGCOLOR](#) is not included in the spec. as yet, but is supported by recent versions of browsers

[BGPROPERTIES](#) allows *watermarking* of documents.

[LEFTMARGIN](#) allows setting of the left margin of the document.

[TOPMARGIN](#) allows setting of the top margin of the document.

The last three attributes are **Internet Explorer** specific.

Background Colour Chart

Background Colours - Example chart

This chart shows a variety of possible colours, together with their #rrgbbb hexadecimal triplet values. All could be used as valid HTML document backgrounds.

RGB TRIPLET COLOR CHART										<i>E-Mail-ware by Doug Jacobson</i>	
FFFFFF	FFFCC	FFF99	FFF66	FFF33	FFF00	FFCCFF	FFCCCC				
FF9933	FF9900	FF66FF	FF66CC	FF6699	FF6666	FF6633	FF6600				
FF0099	FF0066	FF0033	FF0000	CCFFFF	CCFFCC	CCFF99	CCFF66				
CC99FF	CC99CC	CC9999	CC9966	CC9933	CC9900	CC66FF	CC66CC				
CC3333	CC3300	CC00FF	CC00CC	CC0099	CC0066	CC0033	CC0000				
99CC99	99CC66	99CC33	99CC00	9999FF	9999CC	999999	999966				
9933FF	9933CC	993399	993366	993333	993300	9900FF	9900CC				
66FF33	66FF00	66CCFF	66CCCC	66CC99	66CC66	66CC33	66CC00				
666699	666666	666633	666600	6633FF	6633CC	663399	663366				
33FFFF	33FFCC	33FF99	33FF66	33FF33	33FF00	33CCFF	33CCCC				
339933	339900	3366FF	3366CC	336699	336666	336633	336600				
330099	330066	330033	330000	00FFFF	00FFCC	00FF99	00FF66				
0099FF	0099CC	009999	009966	009933	009900	0066FF	0066CC				
003333	003300	0000FF	0000CC	000099	000066	000033	EE0000				
110000	00EE00	00DD00	00BB00	00AA00	008800	007700	005500				
000077	000055	000044	000022	000011	EEEEEE	DDDDDD	BBBBBB				
FFCC99	FFCC66	FFCC33	FFCC00	FF99FF	FF99CC	FF9999	FF9966				
FF33FF	FF33CC	FF3399	FF3366	FF3333	FF3300	FF00FF	FF00CC				
CCFF33	CCFF00	CCCCFF	CCCCCC	CCCC99	CCCC66	CCCC33	CCCC00				
CC6699	CC6666	CC6633	CC6600	CC33FF	CC33CC	CC3399	CC3366				
99FFFF	99FFCC	99FF99	99FF66	99FF33	99FF00	99CCFF	99CCCC				
999933	999900	9966FF	9966CC	996699	996666	996633	996600				
990099	990066	990033	990000	66FFFF	66FFCC	66FF99	66FF66				
6699FF	6699CC	669999	669966	669933	669900	6666FF	6666CC				
663333	663300	6600FF	6600CC	660099	660066	660033	660000				
33CC99	33CC66	33CC33	33CC00	3399FF	3399CC	339999	339966				
3333FF	3333CC	333399	333366	333333	333300	3300FF	3300CC				
00FF33	00FF00	00CCFF	00CCFF	00CC99	00CC66	00CC33	00CC00				
006699	006666	006633	006633	0033FF	0033CC	003399	003366				
DD0000	BB0000	AA0000	AA0000	770000	550000	440000	220000				
004400	002200	001100	001100	0000DD	0000BB	0000AA	000088				
AAAAAA	888888	777777	555555	444444	222222	111111	000000				

If you find this chart useful, please send e-mail to jacobson@phoenix.phoenix.net

NOTE : The colours shown in this chart cannot be guaranteed. Due to resolution/colour depth differences of different Windows video drivers, colours may appear different on either the authors

system or the end users system. This should be considered when using any colour in documents.

<BASE...>

The Base element allows the URL of the document itself to be recorded in situations in which the document may be read out of context. URLs within the document may be in a "partial" form relative to this base address. The <BASE> Element should appear within the bounds of the <HEAD> element.

Where the base address is not specified, the HTML user agent uses the URL it used to access the document to resolve any relative URLs.

The Base element has one attribute, **HREF**, which identifies the URL.

The Netscape Navigator (from version 2.0) adds one other attribute to the **BASE** element. With the introduction of [targeted windows](#), Netscape have added the **TARGET** attribute to the **BASE** element.

This allows you to pick a default named target window for every link in a document that does not have an explicit **TARGET** attribute. It's format is:

```
<BASE TARGET="default_target">
```

NOTE : The use of the **TARGET** attribute is **Netscape** specific.

<ISINDEX...>

The Isindex element tells the HTML user agent that the document is an index document. As well as reading it, the reader may use a keyword search.

The document can be queried with a keyword search by adding a question mark to the end of the document address, followed by a list of keywords separated by plus signs.

NOTE : The Isindex element is usually generated automatically by a server. If added manually to a HTML document, the HTML user agent assumes that the server can handle a search on the document. To use the Isindex element, the server must have a search engine that supports this element.

To the <ISINDEX> element Netscape authors have added the PROMPT attribute. <ISINDEX> indicates that a document is a searchable index.

PROMPT has been added so that text, chosen by the author, can be placed before the text input field of the index. This allows any author chosen message to replace the default text of :

This is a searchable index. Enter search keywords

NOTE : The PROMPT attribute is only supported by **Netscape**.

Another **Netscape** specific attribute is **ACTION**. When used in the <ISINDEX> element, it specifies the cgi script or program to which the text string in the input box should be passed.

For example:

```
<ISINDEX ACTION="websearch">
```

would pass the text entered into the input box on the page to the cgi script "websearch".

NOTE : "websearch" in the above example is a hypothetical cgi script. If used, the ACTION attribute must point to a properly configured script on the host machine. ACTION is **Netscape** specific.

<LINK...>

The Link element indicates a relationship between the document and some other object. A document may have any number of Link elements.

The Link element is empty (does not have a closing element), but takes the same attributes as the [Anchor](#) element.

Typical uses are to indicate authorship, related indexes and glossaries, older or more recent versions, etc. Links can indicate a static tree structure in which the document was authored by pointing to a "parent" and "next" and "previous" document, for example.

Servers may also allow links to be added by those who do not have the right to alter the body of a document.

<NEXTID...>

The Nextid element is a parameter read by and generated by text editing software to create unique identifiers. This element takes a single attribute which is the next document-wide alpha-numeric identifier to be allocated of the form z123 :

```
<NEXTID N=z127>
```

When modifying a document, existing anchor identifiers should not be reused, as these identifiers may be referenced by other documents. Human writers of HTML usually use mnemonic alphabetic identifiers. HTML user agents may ignore the Nextid element. Support for the Nextid element does not impact HTML user agents in any way.

<TITLE> ... </TITLE>

Every HTML document must have a Title element. The title should identify the contents of the document and in a global context, and may be used in history lists and as a label for the windows displaying the document. Unlike headings, titles are not typically rendered in the text of a document itself.

The Title element must occur within the head of the document and may not contain anchors, paragraph elements, or highlighting. Only one title is allowed in a document.

NOTE : The length of a title is not limited, however, long titles may be truncated in some applications. To minimise the possibility, titles should be fewer than 64 characters. Also keep in mind that a short title, such as 'Introduction' may be meaningless out of context. An example of a meaningful title might be 'Introduction to HTML elements'

This is the **only** element that is **required** within the Head element. The other elements described are optional and can be implemented when appropriate

```
<HEAD>  
<TITLE> Introduction to HTML</TITLE>  
</HEAD>
```

<META...>

[Attributes](#)

[See Also](#)

The Meta element is used within the Head element to embed document meta-information not defined by other HTML elements. Such information can be extracted by servers/clients for use in identifying, indexing and cataloguing specialised document meta-information.

Although it is generally preferable to use named elements that have well defined semantics for each type of meta-information, such as title, this element is provided for situations where strict SGML parsing is necessary and the local DTD is not extensible.

In addition, HTTP servers can read the content of the document head to generate response headers corresponding to any elements defining a value for the attribute `HTTP-EQUIV`. This provides document authors a mechanism (not necessarily the preferred one) for identifying information that should be included in the response headers for an HTTP request.

Attributes of the Meta element :

HTTP-EQUIV

This attribute binds the element to an HTTP response header. If the semantics of the HTTP response header named by this attribute is known, then the contents can be processed based on a well-defined syntactic mapping whether or not the DTD includes anything about it. HTTP header names are not case sensitive. If not present, the `NAME` attribute should be used to identify this meta-information and it should not be used within an HTTP response header.

NAME

Meta-information name. If the name attribute is not present, then name can be assumed equal to the value `HTTP-EQUIV`.

CONTENT

The meta-information content to be associated with the given name and/or HTTP response header.

Examples :

If the document contains :

```
<META HTTP-EQUIV="Expires" CONTENT="Tue, 04 Dec 1993 21:29:02 GMT">
<META HTTP-EQUIV="Keywords" CONTENT="Fred, Barney">
<META HTTP-EQUIV="Reply-to" CONTENT="fielding@ics.uci.edu <Roy Fielding">
```

Then the HTTP response header would be :

```
Expires: Tue, 04 Dec 1993 21:29:02 GMT
Keywords: Fred, Barney
Reply-to: fielding@ics.uci.edu (Roy Fielding)
```

When the `HTTP-EQUIV` attribute is not present, the server should not generate an HTTP response header for this meta-information. e.g,

```
<META NAME="IndexType" CONTENT="Service">
```

Do *not* use the Meta element to define information that should be associated with an existing HTML element.

Example of an inappropriate use of the Meta element :

```
<META NAME="Title" CONTENT="The Etymology of Dunsel">
```


Do *not* name an `HTTP-EQUIV` equal to a responsive header that should typically only be generated by the HTTP server. Some inappropriate names are "Server", "Date" and "Last-modified". Whether a name is inappropriate depends on the particular server implementation. It is recommended that servers ignore any Meta elements that specify HTTP-equivalents equal (case-insensitively) to their own reserved response headers.

The `META` element is particularly useful for constructing [Dynamic documents](#)

The following attributes are allowed
within the <META . . .> element.

HTTP-EQUIV

NAME

CONTENT

Dynamic Documents

Block Formatting Elements

Block formatting elements are used for the formatting of whole blocks of text within a HTML document, rather than single characters. They should all (if present) be within the body of the document. The essential block formatting elements are :

<ADDRESS> ... </ADDRESS> - Format an address section
<H1> ... </H1> - Format six levels of heading
<HR> - Renders a sizeable hard line on the page

 - Force a line break
<P> ... </P> - Specify what text constitutes a paragraph and its alignment
<PRE> ... </PRE> - Use text already formatted
<BLOCKQUOTE> ... </BLOCKQUOTE> - To quote text from another source
<CENTER> ... </CENTER>- Centering text on the page.
<NOBR>- Specifying that words aren't to be broken
<WBR>- Specifying that a word is to be broken if necessary
<BASEFONT SIZE=...> - Specifying the default font size for the document.
 ... - Setting/changing the font size, colour and type
<DIV> ... </DIV> - Allow centering, or left/right justification of text.

<ADDRESS> ... </ADDRESS>

The Address element specifies such information as address, signature and authorship, often at the top or bottom of a document.

Typically, an Address is rendered in an italic typeface and may be indented. The Address element implies a paragraph break before and after.

Example of use:

```
<ADDRESS>  
Newsletter editor<BR>  
J.R. Brown<BR>  
JimquickPost News, Jumquick, CT 01234<BR>  
Tel (123) 456 7890  
</ADDRESS>
```



Newsletter editor

J.R. Brown

JimquickPost News, Jimquick, CT 01234

Tel (123) 456 7890

<H1> ... </H1> Headings

Attributes

HTML defines six levels of heading. A Heading element implies all the font changes, paragraph breaks before and after, and white space necessary to render the heading.

The highest level of headings is <H1>, followed by <H2> ... <H6>.

Example of use:

```
<H1>This is a heading</H1>
Here is some text
<H2>Second level heading</H2>
Here is some more text.
```

The rendering of headings is determined by the HTML user agent, but typical renderings are:

```
<H1> ... </H1>
```

Bold, very-large font, centred. One or two blank lines above and below.

```
<H2> ... </H2>
```

Bold, large font, flush-left. One or two blank lines above and below.

```
<H3> ... </H3>
```

Italic, large font, slightly indented from the left margin. One or two blank lines above and below.

```
<H4> ... </H4>
```

Bold, normal font, indented more than H3. One blank line above and below.

```
<H5> ... </H5>
```

Italic, normal font, indented as H4. One blank line above.

```
<H6> ... </H6>
```

Bold, indented same as normal text, more than H5. One blank line above.

Although heading levels can be skipped (for example, from H1 to H3), this practice is discouraged as skipping heading levels may produce unpredictable results when generating other representations from HTML.



Included in the proposed HTML level 3.0 specification is the ability to align Headings.

Basically, **ALIGN**=left|center|right attributes have been added to the <H1> through to <H6> elements. e.g :

```
<H1 ALIGN=center>Hello, this is a heading</H1>
```

would align a heading of style 1 in the centre of the page.

NOTE : This element is currently only supported by Mosaic 2.0 beta 4. Recent versions of Netscape support the <HX ALIGN=CENTER>...</HX> attribute, but not the left/right alignments.



The following attributes are supported by recent browsers under the <HX> element.

ALIGN

NOTE : These Headings are a screenshot of Mosaic 2.0 beta 4 heading rendering, using a default installation. The exact format can be altered within Mosaic, this screenshot is provided to show the six different headings in relation to each other.

This is Heading 1

This is Heading 2

This is Heading 3

This is Heading 4

This is Heading 5

This is Heading 6

Hello, this is a centred
heading

Hello, this is a right aligned
heading

Hello, this is a left aligned
heading

<HR>

Attributes

A Horizontal Rule element is a divider between sections of text such as a full width horizontal rule or equivalent graphic.

Example of use:

```
<HR>  
<ADDRESS>February 8, 1995, CERN</ADDRESS>  
</BODY>
```

The <HR> element specifies that a horizontal rule of some sort (The default being a shaded engraved line) be drawn across the page. To this element recent browsers have added support for 4 new attributes which allow the document author to describe how the horizontal rule should look

<HR **SIZE**=number>

The **SIZE** attribute lets the author give an indication of how thick they wish the horizontal rule to be.

<HR **WIDTH**=number|percent>

The default horizontal rule is always as wide as the page. With the **WIDTH** attribute, the author can specify an exact width in pixels, or a relative width measured in percent of document width.

<HR **ALIGN**=left|right|center>

Now that horizontal rules do not have to be the width of the page it is necessary to allow the author to specify whether they should be pushed up against the left margin, the right margin, or centred in the page.

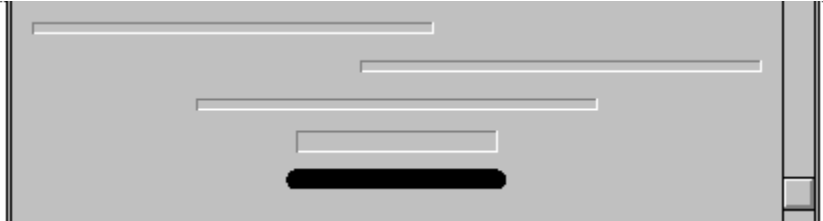
<HR **NOSHADE**>

Finally, for those times when a solid bar is required, the **NOSHADE** attribute lets the author specify that the horizontal rule should not be shaded at all.



The following <HR> render as shown

```
<HR SIZE=5 WIDTH=200 ALIGN=Left>  
<HR SIZE=5 WIDTH=200 ALIGN=Right>  
<HR SIZE=5 WIDTH=200 ALIGN=Center>  
<HR SIZE=10 WIDTH=100 ALIGN=Center>  
<HR SIZE=10 WIDTH=100 ALIGN=Center NOSHADE>
```



NOTE : The page side bars are shown to emphasise the line positioning on the page.

Recent browsers support these additional attributes to the <HR> element

SIZE

WIDTH

ALIGN

NOSHADE

All are detailed on this page

The Line Break element specifies that a new line must be started at the given point. A new line indents the same as that of line-wrapped text.

Example of use:

```
<P>  
Pease porridge hot<BR>  
Pease porridge cold<BR>  
Pease porridge in the pot<BR>  
Nine days old.
```

With the addition of [floating images](#) it was necessary to expand the
 element. Normal
 still just inserts a line break. A CLEAR attribute has been added to
, so :

CLEAR=left will break the line, and move vertically down until you have a clear left margin (no *floating images*).

CLEAR=right does the same for the right margin.

CLEAR=all moves down until both margins are clear of images.

NOTE : The screenshots on the [Element](#) page use <BR CLEAR=left> and <BR CLEAR=right> respectively.

NOTE : The CLEAR attribute (as well as floating images) are currently only supported by Netscape and the Internet Explorer.

<P> ... </P>

The Paragraph element indicates a paragraph. The exact indentation, leading, etc. of a paragraph is not defined and may be a function of other elements, style sheets, etc.

Typically, paragraphs are surrounded by a vertical space of one line or half a line. This is typically not the case within the Address element and or is never the case within the Preformatted Text element. With some HTML user agents, the first line in a paragraph is indented.

Example of use:

```
<H1>This Heading Precedes the Paragraph</H1>
<P>This is the text of the first paragraph.
<P>This is the text of the second paragraph. Although you do not need to start
  paragraphs on new lines, maintaining this convention facilitates document
  maintenance.
<P>This is the text of a third paragraph.
```

Included in the proposed HTML level 3.0 specification is the ability to align paragraphs Basically, **ALIGN**=left|center|right attributes have been added to the <P> element.

e.g :

```
<P ALIGN=LEFT> ... </P>
```

All text within the paragraph will be aligned to the left side of the page layout. This setting is equal to the default <P> element.

```
<P ALIGN=CENTER> ... </P>
```

All text within the paragraph will be aligned to the centre of the page.

```
<P ALIGN=RIGHT> ... </P>
```

All text will be aligned to the right side of the page.

NOTE: To account for the commonly used yet non-standard <CENTER> element, Mosaic (2.0fb) will change the default **ALIGN=LEFT** attribute of all paragraph and header elements to **ALIGN=CENTER** until a **</CENTER>** element is read. Mosaic will also allow internally defined alignment attributes to take precedence over a wrapping **CENTER** element. Mosaic authors would like to encourage all HTML authors to conform to the HTML 3.0 way of centering HTML and no longer use the non-standard **<CENTER>** element.



All of this paragraph
is left
aligned

All of this
paragraph
is centered

All of this
paragraph
is right
aligned

<PRE> ... </PRE>

The Preformatted Text element presents blocks of text in fixed-width font, and so is suitable for text that has been formatted on screen.

The <PRE> element may be used with the optional `WIDTH` attribute, which is a Level 1 feature. The `WIDTH` attribute specifies the maximum number of characters for a line and allows the HTML user agent to select a suitable font and indentation. If the `WIDTH` attribute is not present, a width of 80 characters is assumed. Where the `WIDTH` attribute is supported, widths of 40, 80 and 132 characters should be presented optimally, with other widths being rounded up.

Within preformatted text:

- Line breaks within the text are rendered as a move to the beginning of the next line.
- The <P> element should not be used. If found, it should be rendered as a move to the beginning of the next line.
- Anchor elements and character highlighting elements may be used.
- Elements that define paragraph formatting (headings, address, etc.) must not be used.
- The horizontal tab character (encoded in US-ASCII and ISO-8859-1 as decimal 9) must be interpreted as the smallest positive nonzero number of spaces which will leave the number of characters so far on the line as a multiple of 8. Its use is not recommended however.

NOTE: References to the "beginning of a new line" do not imply that the renderer is forbidden from using a constant left indent for rendering preformatted text. The left indent may be constrained by the width required.

Example of use:

```
<PRE WIDTH="80">
This is an example line.
</PRE>
```

NOTE: Within a Preformatted Text element, the constraint that the rendering must be on a fixed horizontal character pitch may limit or prevent the ability of the HTML user agent to render highlighting elements specially.

<BLOCKQUOTE> ... </BLOCKQUOTE>

The Blockquote element is used to contain text quoted from another source.

A typical rendering might be a slight extra left and right indent, and/or italic font. The Blockquote element causes a paragraph break, and typically provides space above and below the quote.

Single-font rendition may reflect the quotation style of Internet mail by putting a vertical line of graphic characters, such as the greater than symbol (>), in the left margin.

Example of use:

I think the poem ends

```
<BLOCKQUOTE>
```

```
<P>Soft you now, the fair Ophelia. Nymph, in thy orisons, be all my sins  
remembered.
```

```
</BLOCKQUOTE>
```

but I am not sure.



NOTE : This screenshot shows how Netscape 1.1 would display text using the <BLOCKQUOTE> element. Renderings using different HTML user agents may differ.

I think the poem ends

Soft you now, the fair Ophelia.
Nymph, in thy orisons, be all my
sins remembered.

but I am not sure.

<CENTER>

[See Also](#)

All lines of text between the begin and end of the <CENTER> element are centred between the current left and right margins. A new element has been introduced rather than using the proposed [<P ALIGN=CENTER>](#) because using <P ALIGN=CENTER> breaks many existing browsers when the <P> element is used as a container. The <P ALIGN=CENTER> element is also less general and does not support all cases where centering may be desired.

```
<CENTER>All this text would be centred in the page</CENTER>
```

NOTE : Most browsers will internally work-round this element to produce the desired format, but it is an element introduced by Netscape authors.

<P> ... </P> Paragraph Element

<NOBR>

The <NOBR> element stands for **NO B**reak. This means all the text between the start and end of the <NOBR> elements cannot have line breaks inserted. While <NOBR> is essential for those character sequences that don't want to be broken, please be careful; long text strings inside of <NOBR> elements can look rather odd. Especially if during viewing, the user adjust the page size by altering the window size.

NOTE : The <NOBR> Element is supported by Netscape and the Internet Explorer.

<WBR>

The <WBR> element stands for **Word B**reak. This is for the very rare case when a <NOBR> section requires an exact break. Also, it can be used any time the Netscape Navigator can be helped by telling it where a word is allowed to be broken. The <WBR> element does not force a line break (
 does that) it simply lets the Netscape Navigator know where a line break is allowed to be inserted if needed.

NOTE : The <WBR> Element is supported by Netscape and the Internet Explorer.

[Attributes](#)

[See Also](#)

Netscape 1.0 (and above) and Microsoft's Internet Explorer support different sized fonts within HTML documents. This should be distinguished from [Headings](#).

The new element is . Valid values range from 1-7. The default FONT size is 3. The value given to size can optionally have a '+' or '-' character in front of it to specify that it is relative to the document baseFONT. The default baseFONT is 3, and can be changed with the [<BASEFONT SIZE ...>](#) element.

```
<FONT SIZE=4> changes the font size to 4 </FONT>
```

```
<FONT SIZE=+2> changes the font size to <BASEFONT SIZE ...> + 2 </FONT>
```

NOTE : The element is currently not supported by Mosaic.

Microsoft's Internet Explorer supports the ability to change the font colour as well as face type. It adds COLOR and FACE attributes to the element. Netscape will support the use of the COLOR attribute.

```
COLOR = #rrggbb or COLOR = color
```

The colour attribute sets the colour which text will appear in on the screen. Rrggbb is a hexadecimal colour denoting a RGB colour value. Alternately, the colour can be set to one of 16 predefined colours - Black, Olive, Teal, Red, Blue, Maroon, Navy, Gray, Lime, Fuchsia, White, Green, Purple, Silver, Yellow or Aqua. These colour names can be used for the BGCOLOR, TEXT, LINK, and VLINK attributes of the <BODY> tag as well. **NOTE :** The use of names for colouring text is currently only supported by the Microsoft Internet Explorer and Netscape.

Example:

```
<FONT COLOR=#ff0000>This text is red.</FONT>
```

or

```
<FONT COLOR=Red>This text is also red.</FONT>
```



```
FACE=name [,name] [,name]
```

The FACE attribute sets the typeface that will be used to display the text on the screen. The type face displayed must already be installed on the users computer. Substitute type faces can be specified in case the chosen type face is not installed on the customers computer. If no match is found, the text will be displayed in the default type.

Example:

```
<FONT FACE="Arial","Lucida Sans"> This text will be displayed in either Arial, Lucida Sans, or Times Roman, depending on which fonts you have installed on your system.</FONT>
```

NOTE : When using this element, care should be taken to try to use font types that will be installed

on the users computer if you want the text to appear as desired. Changing the font face is **Internet Explorer** specific.

Setting the <BASEFONT> size

The following attributes are supported within the element. **NOTE** : The FACE attribute is specific to Microsoft's Internet Explorer. Both the Internet Explorer and Netscape support the COLOR attribute.

SIZE

COLOR

FACE

The following shows examples of the 16 default colours available for text colouring in Microsoft's Internet Explorer and Netscape. Any other colour can be used, if expressed as an rrggbb hex triplet.

Some Black text	Some Olive text
Some Teal text	Some Red text
Some Blue text	Some Maroon text
Some Navy text	Some Gray text
Some Lime text	Some Fuchsia text
That's white text	
Some Green text	Some Purple text
Some Silver text	Some Yellow text
Some Aqua text	

<BASEFONT SIZE ...>

[See Also](#)

This changes the size of the BASEFONT that all relative [](#) changes are based on. It defaults to 3, and has a valid range of 1-7.

```
<BASEFONT SIZE=5>
```

NOTE : The <BASEFONT SIZE=...> element is **Netscape** specific

Changing the

<DIV> ... </DIV>

[See Also](#)

NOTE : Use of the <DIV> element is currently only supported by **Netscape** (after version 2.0)

The <DIV> element, as described in the HTML 3.0 specification, should be used with a `CLASS` attribute, to name a section of text as being of a certain style. Netscape has implemented the `DIV` element to work as the [<P ALIGN= ...>](#) element. Essentially, text surrounded by the <DIV> ... </DIV> elements will be formatted according to the description attached to the **ALIGN** attribute within the <DIV> elements.

For example :

```
<DIV ALIGN="left">Left justify text by putting it within the DIV tags.</DIV>
```

```
<DIV ALIGN="center">Centre some text by putting it within the DIV tags.</DIV>
```

```
<DIV ALIGN="right">Right justify some text by putting it within the DIV tags.</DIV>
```


<P_ALIGN> - Paragraph Alignment

<A...> ... Anchor

Attributes

An Anchor element is a marked text that is the start and/or destination of a hypertext link. Anchor elements are defined by the <A> element. The <A> element accepts several attributes, but either the NAME or HREF attribute is required.

Attributes of the <A> element :

HREF

If the HREF attribute is present, the text between the opening and closing anchor elements becomes hypertext. If this hypertext is selected by readers, they are moved to another document, or to a different location in the current document, whose network address is defined by the value of the HREF attribute.

Example :

```
See <A HREF="http://www.hal.com/">HaL</A>'s information for more details.
```

In this example, selecting "HaL" takes the reader to a document located at <http://www.hal.com>. The format of the network address is specified in the URI specification for print readers.

With the HREF attribute, the form HREF="#identifier" can refer to another anchor in the same document.

Example :

```
The <A HREF="document.html#glossary">glossary</A> defines terms used in the document.
```

In this example, selecting "glossary" takes the reader to another anchor (i.e. Glossary) in the same document (document.html). The NAME attribute is described below. If the anchor is in another document, the HREF attribute may be relative to the document's address or the specified [base address](#).

Several other forms of the HREF attribute are permitted by browsers. They are as follows :

- Makes a link to another document located on a World Wide Web server.
- Makes a link to an ftp site. Within an HTML document, normally a connection to an anonymous ftp site would be made. The final beta of version 2 of NCSA Mosaic, allows connection to private ftp sites. The Anchor would take the form ftp://jdoe@your.com. However, Mosaic would then prompt the user for a password, so linking from documents to your private ftp site is possibly not a good idea unless youre willing to distribute passwords for the site.
- Makes a link to a gopher server.
- Activating such a link would bring up the browsers mailing dialogue box (providing it has mailing capabilities) allowing the user to send mail messages to the author of the document, or whoevers address follows the mailto attribute. Final beta of NCSA Mosaic supports the **TITLE** sub-attribute to this attribute. It allows the author to specify the subject of the mail message that will be sent.
- Makes a link to a newsgroup. Care should be taken in using such links because the author can not know what newsgroups are carried by the local news server of the user.
- Makes a link to a specific newsrsc file.

- `` Can be used to specify a different news server to that which the user may normally use.
- `` Activating such a link would initiate a telnet session (using an external application) to the machine specified after the telnet:// label.
- `` Makes a link that connects to a specified WAIS index server.

NAME

If present, the `NAME` attribute allows the anchor to be the target of a link. The value of the `NAME` attribute is an identifier for the anchor. Identifiers are arbitrary strings but must be unique within the HTML document.

Example of use:

```
<A NAME=coffee>Coffee</A> is an exmple of...  
An example of this is <A HREF=#coffee>coffee</A>.
```

Another document can then make a reference explicitly to this anchor by putting the identifier after the address, separated by a hash sign :

```
<A NAME=drinks.html#coffee>
```

TITLE

The `Title` attribute is informational only. If present, the `Title` attribute should provide the title of the document whose address is given by the `HREF` attribute.

The `Title` attribute is useful for at least two reasons. The HTML user agent may display the title of the document prior to retrieving it, for example, as a margin note or on a small box while the mouse is over the anchor, or while the document is being loaded. Another reason is that documents that are not marked up text, such as graphics, plain text and Gopher menus, do not have titles. The `TITLE` attribute can be used to provide a title to such documents. When using the `TITLE` attribute, the title should be valid and unique for the destination document.

REL

The `REL` attribute gives the relationship(s) described by the hypertext link from the anchor to the target. The value is a comma-separated list of relationship values. Values and their semantics will be registered by the HTML registration authority. The default relationship if none other is given is void. The `REL` attribute is only used when the `HREF` attribute is present.

REV

The `REV` attribute is the same as the `REL` attribute, but the semantics of the link type are in the reverse direction. A link from A to B with `REL="X"` expresses the same relationship as a link from B to A with `REV="X"`. An anchor may have both `REL` and `REV` attributes.

URN

If present, the `URN` attribute specifies a uniform resource name (URN) for a target document. The format of URNs is under discussion (1994) by various working groups of the Internet Engineering Task Force.

METHODS

The `METHODS` attributes of anchors and links provide information about the functions that the user may perform on an object. These are more accurately given by the HTTP protocol when it is used, but it may, for similar reasons as for the `TITLE` attribute, be useful to include the information in advance in the link. For example, the HTML user agent may chose a different rendering as a function of the methods allowed; for example, something that is searchable may get a different icon.

The value of the `METHODS` attribute is a comma separated list of HTTP methods supported by the object for public use.

TARGET

Browser windows can now have names associated with them. Links in any window can refer to another window by name. When you click on the link, the document you asked for will appear in that named window. If the window is not already open, Netscape will open and name a new window for you.

The syntax for the targeted windows is:

```
<A HREF="url.html" TARGET="window_name">Click here and open a New Window</A>
```

NOTE : The use of targeted browser windows is **Netscape** specific. Also, if the targetted document is part of a frameset, there are various reserved names for target window available for smooth window transition.

See Also, <BASE TARGET= ...>

The following attributes are all allowed within the <A . . . > element

href

name

title

rel

rev

urn

methods

Mailto : title

target - For opening a target window

List Elements

HTML supports several types of lists, all of which may be nested. If used they should be present in the body of a HTML document.

<DL> ... </DL> - Definition list.
<DIR> ... </DIR> - Directory list
<MENU> ... </MENU> - Menu list
 ... - Ordered list
 ... - Unordered list

<DL> ... </DL>

[Attributes](#)

[See Also](#)

A definition list is a list of terms and corresponding definitions. Definition lists are typically formatted with the term flush-left and the definition, formatted paragraph style, indented after the term.

Example of use:

```
<DL>
<DT>Term<DD>This is the definition of the first term.
<DT>Term<DD>This is the definition of the second term.
</DL>
```



If the <DT> term does not fit in the <DT> column (one third of the display area), it may be extended across the page with the <DD> section moved to the next line, or it may be wrapped onto successive lines of the left hand column.

Single occurrences of a <DT> element without a subsequent <DD> element are allowed, and have the same significance as if the <DD> element had been present with no text.

The opening list element must be <DL> and must be immediately followed by the first term (<DT>).

The definition list type can take the **COMPACT** attribute, which suggests that a compact rendering be used, because the list items are small and/or the entire list is large.

Unless you provide the **COMPACT** attribute, the HTML user agent may leave white space between successive <DT>, <DD> pairs. The **COMPACT** attribute may also reduce the width of the left-hand (<DT>) column.

If using the **COMPACT** attribute, the opening list element must be <DL COMPACT>, which must be immediately followed by the first <DT> element:

```
<DL COMPACT>
<DT>Term<DD>This is the first definition in compact format.
<DT>Term<DD>This is the second definition in compact format.
</DL>
```





Term This is the first definition in compact format.

Term This is the second definition in compact format.

The following are all allowed
within the <DL> element.

<DT> : Definition list Term

<DD> : Definition list definition.

COMPACT

<DIR> : Directory list

<MENU> : Menu list

 : Ordered list

 : Unordered list

<DIR> ... </DIR>

[See Also](#)

A Directory List element is used to present a list of items containing up to 20 characters each. Items in a directory list may be arranged in columns, typically 24 characters wide. If the HTML user agent can optimise the column width as function of the widths of individual elements, so much the better.

A directory list must begin with the <DIR> element which is immediately followed by a (list item) element:

```
<DIR>  
<LI>A-H<LI>I-M  
<LI>M-R<LI>S-Z  
</DIR>
```



- A-H
- I-M
- M-R
- S-Z

<DL> : Definition list
<MENU> : Menu list
 : Ordered list
 : Unordered list

<MENU> ... </MENU>

[See Also](#)

A menu list is a list of items with typically one line per item. The menu list style is more compact than the style of an unordered list.

A menu list must begin with a <MENU> element which is immediately followed by a (list item) element:

```
<MENU>  
<LI>First item in the list.  
<LI>Second item in the list.  
<LI>Third item in the list.  
</MENU>
```



- First item in the list.
- Second item in the list.
- Third item in the list.

<DL> : Definition list
<DIR> : Directory list
 : Ordered list
 : Unordered list

 ...

[Attributes](#)

[See Also](#)

The Ordered List element is used to present a numbered list of items, sorted by sequence or order of importance.

An ordered list must begin with the element which is immediately followed by a (list item) element:

```
<OL>
<LI>Click the Web button to open the Open the URL window.
<LI>Enter the URL number in the text field of the Open URL window. The Web
    document you specified is displayed.
<LI>Click highlighted text to move from one link to another.
</OL>
```

The Ordered List element can take the COMPACT attribute, which suggests that a compact rendering be used.



The average ordered list counts 1, 2, 3, ... etc. The **TYPE** attribute has been added to this element to allow authors to specify whether the list items should be marked with:

- (TYPE=A) - capital letters. e.g. A, B, C ...
- (TYPE=a) - small letters. e.g. a, b, c ...
- (TYPE=I) - large roman numerals. e.g. I, II, III ...
- (TYPE=i) - small roman numerals. e.g. i, ii, iii ...
- (TYPE=1) - or the default numbers. e.g. 1, 2, 3 ...

For lists that wish to start at values other than 1 the new attribute **START** is available.

START is always specified in the default numbers, and will be converted based on TYPE before display. Thus START=5 would display either an 'E', 'e', 'V', 'v', or '5' based on the TYPE attribute.



To give even more flexibility to lists, the **TYPE** attribute has been added to the element as well. It takes the same values as and it changes the list type for that item, and all subsequent items. For ordered lists we have also added the **VALUE** element so you can change the count, for that list item and all subsequent.

NOTE : The TYPE attribute used in the Element and the Element and the START attribute in the Element are supported only by Netscape and the Internet Explorer.

1. Click the Web button to open the Open the URL window.
2. Enter the URL number in the text field of the Open URL window. The Web document you specified is displayed.
3. Click highlighted text to move from one link to another.

<DL> : Definition list
<DIR> : Directory list
<MENU> : Menu list
 : Unordered list

The following Ordered List, was rendered using `TYPE=a START=3`

- c. Click the Web button to open the Open the URL window.
- d. Enter the URL number in the text field of the Open URL window.
The Web document you specified is displayed.
- e. Click highlighted text to move from one link to another.

The following attributes are allowed in the
 element.

TYPE

START

VALUE

 ...

[See Also](#)

The Unordered List element is used to present a list of items which is typically separated by white space and/or marked by bullets.

An unordered list must begin with the element which is immediately followed by a (list item) element:

```
<UL>
<LI>First list item
<LI>Second list item
<LI>Third list item
</UL>
```



The Unordered List element can take the COMPACT attribute, which suggests that a compact rendering be used.

The basic bulleted list has a default progression of bullet types that changes as you move through indented levels. From a solid disc, to a circle to a square. **Netscape** authors have added a **TYPE** attribute to the element so that no matter what the indent level the bullet type can be specified thus :

```
TYPE=disc
TYPE=circle
TYPE=square
```



To give even more flexibility to lists, **Netscape** authors have added the **TYPE** attribute to the element as well. It takes the same values as and it changes the list type for that item, and all subsequent items.

NOTE : The TYPE attribute when used in the and Elements is Netscape specific.

- First list item
- Second list item
- Third list item

<DL> : Definition list
<DIR> : Directory list
<MENU> : Menu list
 : Ordered list

NOTE : The types disc and circle appear the same when rendered.

TYPE=disc/circle:

- First list item
- Second list item
- Third list item

TYPE=square:

- First list item
- Second list item
- Third list item

Information Type and Character Formatting Elements

The following information type and character formatting elements are by currently available browsers.

Information type elements:

(**NOTE** : Different information type elements may be rendered in the same way)

<CITE> ... </CITE> - Citation
<CODE> ... </CODE> - An example of Code
 ... - Emphasis
<KBD> ... </KBD> - User typed text
<SAMP> ... </SAMP> - A sequence of literal characters
 ... - Strong typographic emphasis
<VAR> ... </VAR> - Indicates a variable name
<!-- ... --> - Defining comments.

Character formatting elements

 ... - Boldface type
<I> ... </I> - Italics
<TT> ... </TT> - TypeType (or Teletype)
<U> ... </U> - Underlined text
<STRIKE> ... </STRIKE> - Text that has been struckthrough
_{...} - Subscript
^{...} - Superscript
<BIG> ... </BIG> - Big text
<SMALL> ... </SMALL> - Small text
<BLINK> ... </BLINK> - Blinking text

Character-level elements are used to specify either the logical meaning or the physical appearance of marked text without causing a paragraph break. Like most other elements, character-level elements include both opening and closing elements. Only the characters between the elements are affected:

```
This is <EM>emphasized</EM> text.
```

Character-level elements may be ignored by minimal HTML applications.

Character-level elements are interpreted from left to right as they appear in the flow of text. Level 1 HTML user agents must render highlighted text distinctly from plain text. Additionally, content must be rendered as distinct from content, and content must rendered as distinct from <I> content.

Character-level elements may be nested within the content of other character-level elements; however, HTML user agents are not required to render nested character-level elements distinctly from non-nested elements:

```
plain <B>bold <I>italic</I></B>
```

may be rendered the same as

```
plain <B>bold </B><I>italic</I>
```

Note that typical renderings for information type elements vary between applications. If a specific rendering is necessary, for example, when referring to a specific text attribute as in "The *italic parts* are mandatory", a formatting element can be used to ensure that the intended rendering is used where possible.

<CITE> ... </CITE>

The Citation element specifies a citation; typically rendered as italics.

This sentence, containing a <CITE>citation reference</CITE>

would look like:

This sentence, containing a *citation reference*

<CODE> ... </CODE>

The Code element indicates an example of code; typically rendered as monospaced . Do not confuse with the [Preformatted Text](#) element.

This sentence contains an <CODE>example of code</CODE>.

It would look like :

This sentence contains an example of code

** ... **

The Emphasis element indicates typographic emphasis, typically rendered as italics.

The `Emphasis` element typically renders as Italics.

would render :

The *Emphasis* element typically render as Italics.

<KBD> ... </KBD>

The Keyboard element indicates text typed by a user; typically rendered as monospaced . It might commonly be used in an instruction manual.

The text inside the <KBD>KBD element, would typically</KBD> render as monospaced font.

would render as :

The text inside the KBD element would typically render as monospaced font.

<SAMP> ... </SAMP>

The Sample element indicates a sequence of literal characters; typically rendered as monospaced.

A sequence of `<SAMP>literal characters</SAMP>` commonly renders as monospaced font.

would render as :

A sequence of `literal characters` commonly renders as monospaced font.

** ... **

The Strong element indicates strong typographic emphasis, typically rendered in bold.

The instructions `must be read` before continuing.

would be rendered as :

The instructions **must be read** before continuing.

<VAR> ... </VAR>

The Variable element indicates a variable name; typically rendered as italic.

When coding, `<VAR>LeftIndent()</VAR>` must be a variable

would render as :

When coding *LeftIndent()* must be a variable.

<!-- Comments -->

To include comments in an HTML document that will be ignored by the HTML user agent, surround them with <!-- and -->. After the comment delimiter, all text up to the next occurrence of --> is ignored. Hence comments cannot be nested. White space is allowed between the closing -- and >, but not between the opening <! and --.

For example:

```
<HEAD>  
<TITLE>HTML Guide: Recommended Usage</TITLE>  
<!-- Id: Text.html,v 1.6 1994/04/25 17:33:48 connolly Exp -->  
</HEAD>
```

NOTE: Some historical HTML user agents incorrectly consider a > sign to terminate a comment.

** ... **

The Bold element specifies that the text should be rendered in boldface, where available. Otherwise, alternative mapping is allowed.

The instructions `must be read` before continuing.

would be rendered as :

The instructions **must be read** before continuing.

<I> ... </I>

The Italic element specifies that the text should be rendered in italic font, where available. Otherwise, alternative mapping is allowed.

Anything between the <I>I elements</I> should be italics.

would render as :

Anything between the *I elements* should be italics.

<TT> ... </TT>

The Teletype element specifies that the text should be rendered in fixed-width typewriter font.

Text between the <TT> typetype elements</TT> should be rendered in fixed width typewriter font.

would render as :

Text between the typetype elements should be rendered in fixed width typewriter font.

<U> ... </U>

The `<U> ... </U>` Elements state that the enclosed text should be rendered, if practical, underlined. This is an HTML 3.0 element and may not be widely supported.

The `<U>main point</U>` of the exercise...

would be rendered as :

The main point of the exercise...

NOTE : As yet, Netscape doesn't support use of the `<U>` element. The final beta of Mosaic version 2.0 and the Microsoft Internet Explorer however, do.

<STRIKE> ... </STRIKE>

The <STRIKE> . . . </STRIKE> element states that the enclosed text should be displayed with a horizontal line striking through the text. Alternative mappings are allowed if this is not practical. This is an HTML 3.0 element and may not be widely supported.

This text would be <STRIKE>struck through</STRIKE>

would be rendered as :

This text would be ~~struck through~~

NOTE : While use of the <STRIKE> element is currently supported by Netscape and Mosaic, the element contained in current versions of the HTML 3.0 specification, is <S> . . . </S>, which is supported by Mosaic, but not Netscape. The Microsoft Internet Explorer supports either version of the element.

_{...}

The <SUB> element specifies that the enclosed text should be displayed as a subscript, and if practical, using a smaller font (compared with normal text). This is an HTML 3.0 element and may not be widely supported.

This is the main text, with _{this bit} being subscript.



NOTE : Both Netscape and Mosaic will make the selected text a subscript to the main text, formatting the selected text slightly smaller than the normal text. Netscape can be forced to make subscripts even smaller by compounding the _{...} element with the <SMALL> ... </SMALL> element, or be forced to render the subscript the same size as the normal text, by compounding the _{...} element with the <BIG> ... </BIG> element. The exact appearance of the subscript text (in Netscape) will change depending on any and <BASEFONT SIZE=...> settings, if specified.

This is the main text, with _{this bit} being subscript

^{...}

The <SUP> element specifies that the enclosed text should be displayed as a superscript, and if practical, using a smaller font (compared with normal text). This is an HTML 3.0 element and may not be widely supported.

This is the main text, with ^{this bit} being superscript.



NOTE : Both Netscape and Mosaic will make the selected text a superscript to the main text, formatting the selected text slightly smaller than the normal text. Netscape can be forced to make superscripts even smaller by compounding the ^{...} element with the <SMALL> ... </SMALL> element, or be forced to render the superscript the same size as the normal text, by compounding the ^{...} element with the <BIG> ... </BIG> element.

The exact appearance of the superscript text (in Netscape) will change depending on any and <BASEFONT SIZE=...> settings, if specified.

This is the main text, with ^{this bit} being superscript

<BIG> ... </BIG>

The <BIG> element specifies that the enclosed text should be displayed, if practical, using a big font (compared with the current font). This is an HTML 3.0 element and may not be widely supported.

This is normal text, with <BIG>this bit</BIG> being big text.

would be rendered as:

This is normal text, with **this bit** being big text.

NOTE : Use of this element is currently supported by Netscape only. Netscape also allows the <BIG> ... </BIG> element to be used surrounding the _{...} and ^{...} elements to force rendering of the sub/superscript text as normal size text as opposed to the default slightly smaller text normally used.

The exact appearance of the big text (in Netscape) will change depending on any and <BASEFONT SIZE=...> settings, if specified.

<SMALL> ... </SMALL>

The <SMALL> element specifies that the enclosed text should be displayed, if practical, using a small font (compared with the current font). This is an HTML 3.0 element and may not be widely supported.

This is normal text, with <SMALL>this bit</SMALL> being small text.

would be rendered as:

This is normal text, with this bit being small text.

NOTE : Use of this element is currently supported by Netscape only. Netscape also allows the <SMALL> ... </SMALL> element to be used surrounding the _{...} and ^{...} elements to force rendering of the sub/superscript text as text even smaller than the default slightly smaller (compared to the normal) text normally used. The exact appearance of the small text (in Netscape) will change depending on any and <BASEFONT SIZE=...> settings, if specified.

<BLINK>

Surrounding any text with this element will cause the selected text to *blink* on the viewing page. This can serve to add extra emphasis to selected text.

```
<BLINK>This text would blink on the page</BLINK>
```

NOTE : The <BLINK> ... </BLINK> element is currently only supported by the Netscape Navigator.

In line Images

Recently, the [](#) element has undergone the largest enhancements of all HTML 2.0 elements, on the way to HTML 3.0 being fully standardised and adopted.

The attributes commonly supported by the IMG element have had some recent additions to allow [Client side Image Maps](#), embedded [In-line video](#) clips and also embedded [In-line VRML worlds](#).

<IMG...> In-line images

[Attributes](#)

[See Also](#)

The Image element is used to incorporate in-line graphics (typically icons or small graphics) into an HTML document. This element cannot be used for embedding other HTML text.

HTML user agents that cannot render in-line images ignore the Image element unless it contains the ALT attribute. Note that some HTML user agents can render linked graphics but not in-line graphics. If a graphic is essential, you may want to create a link to it rather than to put it in-line. If the graphic is not essential, then the Image element is appropriate.

The Image element, which is empty (no closing element), has these attributes:

ALIGN

The ALIGN attribute accepts the values TOP or MIDDLE or BOTTOM, which specifies if the following line of text is aligned with the top, middle, or bottom of the graphic.



ALT

Optional text as an alternative to the graphic for rendering in non-graphical environments. Alternate text should be provided whenever the graphic is not rendered. Alternate text is mandatory for Level 0 documents. Example of use:

e.g. : ` Be sure to read these instructions.`

ISMAP

The ISMAP (is map) attribute identifies an image as an image map. Image maps are graphics in which certain regions are mapped to URLs. By clicking on different regions, different resources can be accessed from the same graphic. Example of use:

e.g. : `
`

NOTE : To be able to employ image maps in HTML documents, the HTTP server which will be controlling document access must have the correct cgi-bin software installed to control image map behaviour. i.e. the document must have access to an image map handling script which is pointed to your .map file defining the graphics hot-spots

Microsofts Internet Explorer allows a simpler form of image map, known as client-side image maps. Currently, this type of map is a proposed extension to HTML. For details, see [Client Side Image Maps](#).

SRC

The value of the SRC attribute is the URL of the document to be embedded; only images can be embedded, not HTML text. Its syntax is the same as that of the HREF attribute of the [<A>](#) element. SRC is mandatory. Image elements are allowed within anchors.

Example of use:

e.g. : `Be sure to read these instructions.`

Netscape specific extensions to the Element

e.g. : `<IMG ALIGN= left|right|top|texttop|middle|
absmiddle|baseline|bottom|absbottom>`

The additions to the `ALIGN` options needs a lot of explanation. First, the values "left" and "right". Images with those alignments are an entirely new *floating* image type.

`ALIGN=left` image will float the image down and over to the left margin (into the next available space there), and subsequent text will wrap around the right hand side of that image.

`ALIGN=right` will align the image aligns with the right margin, and the text wraps around the left.



`ALIGN=top` aligns itself with the top of the tallest item in the line.

`ALIGN=texttop` aligns itself with the top of the tallest text in the line (this is usually but not always the same as `ALIGN=top`).

`ALIGN=middle` aligns the baseline of the current line with the middle of the image.

`ALIGN=absmiddle` aligns the middle of the current line with the middle of the image.

`ALIGN=baseline` aligns the bottom of the image with the baseline of the current line.

`ALIGN=bottom` aligns the bottom of the image with the baseline of the current line.

`ALIGN=absbottom` aligns the bottom of the image with the bottom of the current line.

``

The `WIDTH` and `HEIGHT` attributes were added to `` mainly to speed up display of the document. If the author specifies these, the viewer of their document will not have to wait for the image to be loaded over the network and its size calculated. **Netscape** is the only browser that will scale the whole image if either the `WIDTH` or `HEIGHT` attributes are specified, maintaining the aspect ratio. If both are specified then the image is specified accordingly.

``

This lets the document author control the thickness of the border around an image displayed.

Warning: setting `BORDER=0` on images that are also part of anchors may confuse your users as they are used to a coloured border indicating an image is an anchor.

``

For the *floating* images it is likely that the author does not want them pressing up against the text wrapped around the image. `VSPACE` controls the vertical space above and below the image, while `HSPACE` controls the horizontal space to the left and right of the image.

LOWSRC

Using the `LOWSRC` attribute, it is possible to use two images in the same space. The syntax is :

e.g. : ``

Browsers that do not recognise the `LOWSRC` attribute cleanly ignore it and simply load the image called "highres.gif".

The Netscape Navigator, on the other hand, will load the image called "lowres.jpg" on its first layout pass through the document. Then, when the document and all of its images are fully loaded, the Netscape Navigator will do a second pass through and load the image called "highres.gif" in place. This means that you can have a very low-resolution version of an image loaded initially; if the user stays on the page after the initial layout phase, a higher-resolution (and presumably bigger) version of the same image

can "fade in" and replace it.

Both GIF (both normal and interlaced) and JPEG images can be freely interchanged using this method. You can also specify width and/or height values in the `IMG` element, and both the high-res and low-res versions of the image will be appropriately scaled to match.

If the images are of different sizes and a fixed height and width are not specified in the `IMG` element, the second image (the image specified by the `SRC` attribute) will be scaled to the dimensions of the first (`LOWSRC`) image. This is because by the time the Netscape Navigator knows the dimensions of the second image, the first image has already been displayed in the document at its normal dimensions.

The following attributes are allowed
within the element

ALIGN

ALT

ISMAP

SRC

WIDTH

HEIGHT

BORDER

VSPACE

HSPACE

*LOWSRC

LOWSRC is a Netscape specific
extensions.



This text is aligned at the "top" of the picture



This text is aligned at the "middle" of the picture



This text is aligned at the "bottom" of the picture



This picture is aligned to the left of the page. All of this text will fall neatly to the side of the image.

This picture is aligned to the right of the page. All of this text will fall neatly to the side of the image. This image also has a BORDER of 2



[Client side Image maps](#)
[In-line video](#)

Client Side Image Maps

[Attributes](#)

[See Also](#)

NOTE : Currently, Client Side Image maps are supported by Microsoft's Internet Explorer, all SpyGlass browsers that include licensed SpyGlass technology (such as Enhanced Mosaic), Netscape and Mosaic.

The current technique for implementing image maps requires communication with an HTTP server to process co-ordinate information and generate a new URL. This element allows browsers to process image maps internally. These extensions allow the use of image maps in local HTML files or files accessed via alternate transport mechanisms such as FTP. They also allow image map characteristics to be specified in a format that is not specific to the server software.

Syntax

Adding a **USEMAP** attribute to an [](#) element indicates that it is a client-side image map. The **USEMAP** attribute can be used with the [ISMAP](#) attribute to indicate that the image can be processed as either a client-side or server-side image map. The argument to **USEMAP** specifies which map to use with the image, in a format similar to the **HREF** attribute on anchors. If the argument to **USEMAP** starts with a '#', it is assumed to be in the same document as the **IMG** tag.

A few examples would be:

This method would only work if your browser supports client-side image maps:

```
<IMG SRC="../images/tech/pic1.gif" USEMAP="maps.html#map1">
```

This image map will work regardless, because it specifies a server side map file as well as a client side image map file.

```
<A HREF="/cgi-bin/image map/pic2"><IMG SRC="../images/tech/pic2.gif"
USEMAP="maps.html#map2" ISMAP></A>
```

Clicking here will take you to a page with an error message if you don't have client-side image map support:

```
<A HREF="no_csim.html"><IMG SRC="../images/tech/pic3.gif"
USEMAP="maps.html#map3"></A>
```

The different regions of the image are described using a **MAP** element. The map describes each region in the image and indicates where it links to. The basic format for the **MAP** element is as follows:

```
<MAP NAME="name">
<AREA [SHAPE="shape"] COORDS="x,y,..." [HREF="reference"] [NOHREF]>
</MAP>
```

The **MAP** definition can reside in the same file as the image that will use it, or in a completely separate file. This way, if you intend to use this type of image map extensively, all the map definitions can be kept separate from the main documents, allowing easier maintenance.

The name specifies the name of the map so that it can be referenced by an [](#) element. The shape gives the shape of this area. Currently the only shape defined is "RECT", but the syntax is defined in such a way to allow other region types to be added. If the **SHAPE** tag is omitted, **SHAPE="RECT"** is assumed. The **COORDS** tag gives the co-ordinates of the shape, using image pixels as the units. For a rectangle, the coordinates are given as "left,top,right,bottom". The rectangular region defined includes the lower-right corner specified, i.e. to specify the entire area of a 100x100 image, the co-ordinates would be "0,0,99,99".

The `NOHREF` tag indicates that clicks in this region should perform no action. An `HREF` tag specifies where a click in that area should lead. Note that a relative anchor specification will be expanded using the URL of the map description as a base, rather than using the URL of the document from which the map description is referenced. If a `BASE` tag is present in the document containing the map description, that URL will be used as the base.

An arbitrary number of `AREA` tags may be specified. If two areas intersect, the one which appears first in the map definition takes precedence in the overlapping region. For example, a button bar in a document might use a 160 pixel by 60 pixel image and appear like this:

```
<MAP NAME="buttonbar">
<AREA SHAPE="RECT" COORDS="10,10,49,49" HREF="about_us.html">
<AREA SHAPE="RECT" COORDS="60,10,99,49" HREF="products.html">
<AREA SHAPE="RECT" COORDS="110,10,149,49" HREF="index.html">
<AREA SHAPE="RECT" COORDS="0,0,159,59" NOHREF>
</MAP>
<IMG SRC="../images/tech/bar.gif" USEMAP="#buttonbar">
```

This example includes a region encompassing the entire image with a `NOHREF` tag, but this is actually redundant. Any region of the image that is not defined by an `AREA` tag is assumed to be `NOHREF`.

This syntax provides maximum flexibility to the document author for dealing with browsers which do not support this extension, since such browsers will ignore the `MAP` and `AREA` elements. If the document resides on an HTTP server, the server can still provide `ISMAP`-style support. Otherwise, the author can choose to have the image not appear as an anchor at all, or have a click anywhere within it lead to another page, perhaps providing an equivalent textual list of options.

Because the map description can reside in a different file additional flexibility is provided. A common use of image maps is a button bar which appears at the bottom of every document. The map description could be specified in one file, such as the server's home page, and referenced from each document. Thus, the map could be modified by changing a single map description rather than having to modify every file on the server. There is also the possibility of advanced applications with servers dynamically generating map descriptions, similar to the way that some servers currently dynamically generate image files.

The following attributes are supported when using Client Side Image maps. They are all to be included within the bounds of a normal IMG Element

USEMAP

MAP

SHAPE

COORDS

AREA

 Element
ISMAP attribute

In-line Video

[Attributes](#)

[See Also](#)

Microsoft's Internet Explorer allows the user to embed .AVI (Audio Video Interleave) video clips in HTML documents. This is done by adding several new attributes, notably `DYNSRC` (Dynamic Source) to the `` element. Using the `IMG` element for this purpose makes it possible to add video clips to pages, but also have non video enabled browsers display still images in their place.

```
<IMG DYNSRC="video.avi" SRC="TheEarth.gif" WIDTH=50 HEIGHT=50 LOOP=INFINITE
  ALIGN=RIGHT>
```

DYNSRC

Specifies the address of a video clip to be displayed in the window. It stands for **Dynamic Source**.

e.g. ``

If your browser supports inline video, you will see the movie "bar.avi"; otherwise you will see the picture "foo.gif"

START

For video clips, specifies when the file should start playing. This attribute can be set to `FILEOPEN` or `MOUSEOVER`. `FILEOPEN` means that the video will start playing as soon as it has finished opening. This is the default. `MOUSEOVER` means start playing when the user moves the mouse cursor over the animation. Both can be specified together if necessary.

CONTROLS

For video clips, if present, a set of controls is displayed under the clip window. This allows the user to pause/restart or skip sections of the video if desired. It is a flag and has no sub-attributes.

LOOP

Specifies how many times a video clip will loop when activated. If `n=-1`, or if `LOOP=INFINITE` is specified, the video will loop indefinitely.

LOOPDELAY

Specifies, in milliseconds, how long a video clip will wait between play loops.

NOTE : As seen in the first example on this page, because the `DYNSRC` is an attribute of the `IMG` element, other attributes of the `IMG` element, such as `HEIGHT` and `WIDTH` will also apply to the in-line video, specifying the size of the video window.

Examples.

```
<IMG SRC="preview3.gif" DYNSRC="windsurfing.avi" START=MOUSEOVER,FILEOPEN><BR>
```

The above video will play once as soon as it opens and thereafter will play whenever the user's mouse moves over the video window. For browsers that do not support in-line video, the graphic file "preview3.gif" will be displayed.

```
<IMG SRC="preview1.gif" DYNSRC="skiing.avi" START=FILEOPEN LOOPDELAY=1000
  LOOP=5><BR>
```

The above video will start one second after it is done loading, then loop five times, then stop.

The following attributes can be perceived as sub-attributes of the DYNSRC attribute, the main attribute of the IMG element that allows embedding of video clips.

START

CONTROLS

LOOP

LOOPDELAY

 Element

<EMBED> Used for OLE object embedding

The **Virtual Reality Modelling Language** (VRML) is a language for describing multi-participant interactive simulations - virtual worlds networked via the global Internet and hyperlinked with the World Wide Web. All aspects of the virtual world display, interaction and internetworking can be specified using VRML.

In-line VRML Worlds

Microsoft's **Internet Explorer** (from version 2) has added the ability to include in-line embedded [VRML](#) worlds by installing the **Virtual Explorer** plug-in module, available from the Microsoft Windows95 Web site (<http://www.windows.microsoft.com>). It does this by adding the **VRML** attribute to the [](#) element.

As the attribute is used in the [](#) element, it supports many of the other attributes of the [](#) element, such as **HEIGHT** and **WIDTH**.

As well as [.WRL](#) VRML virtual world files, using the Virtual Explorer, [.XAF](#) [ActiveVRML](#) files can be embedded into HTML documents. These allow for animated VRML objects that can be freely manipulated in 3-D space and may respond to user events, or change with time. As such, Microsoft's ActiveVRML specification represents a possible major enhancement to what may become VRML 2.0.

For example;

```
<IMG SRC="picture.gif" VRML="world.wrl" HEIGHT=250 WIDTH=300>
```

The above example, would embed the VRML world, "world.wrl" into the HTML document, with the navigation controls below the embedding pane. The pane is displayed according to the dimensions specified. For browsers, other than the **Virtual Explorer** (Internet Explorer with the VRML add-on), the picture "picture.gif" would be displayed.

NOTE : Embedding of VRML worlds is also supported by Netscape, using the WebFX [plug-in module](#) and the [<EMBED>](#) element.

ActiveVRML is at present a draft specification, available from Microsoft at <http://www.microsoft.com/intdev/inttech/rbintro.htm>, or from the Internet Explorer web site, <http://www.microsoft.com/windows/ie/> The specification details additions to the [VRML](#) language that allows the creation of interactive animations. Using ActiveVRML, the author can create animations by describing how animation parameters vary continuously with time, user input and other parameters, instead of creating conventional frames to make an animation. That is, the author describes events influencing objects, which results in other events happening, leading to animation.

Forms

The inclusion of the Form elements, allowing user input/feedback on HTML documents, was the major difference between the HTML specification 2.0 and its predecessors.

They are created by placing input fields within paragraphs, preformatted/literal text and lists. This gives considerable flexibility in designing the layout of forms.

The following elements are used to create forms :

<FORM> ... </FORM> - A form within a document
<INPUT ...> ... </INPUT> - One input field
<OPTION> - One option within a Select element.
<SELECT> ... <SELECT> - A selection from a finite set of options
<TEXTAREA ...> ... </TEXTAREA> - A multi-line input field

Each variable field is defined by an INPUT, TEXTAREA, or OPTION element and must have a NAME attribute to identify its value in the data returned when the form is submitted.

Example of use (a questionnaire form):

```
<H3>Sample Questionnaire</H3>
<P>Please fill out this questionnaire:
<FORM METHOD="POST" ACTION="http://www.hal.com/sample">
<P>Your name: <INPUT NAME="name" size="48">
<P>Male <INPUT NAME="gender" TYPE=RADIO VALUE="male">
<P>Female <INPUT NAME="gender" TYPE=RADIO VALUE="female">
<P>Number in family: <INPUT NAME="family" TYPE=text>
<P>Cities in which you maintain a residence:
<UL>
<LI>Kent <INPUT NAME="city" TYPE=checkbox VALUE="kent">
<LI>Miami <INPUT NAME="city" TYPE=checkbox VALUE="miami">
<LI>Other <TEXTAREA NAME="other" cols=48 rows=4></textarea>
</UL>
Nickname: <INPUT NAME="nickname" SIZE="42">
<P>Thank you for responding to this questionnaire.
<P><INPUT TYPE=SUBMIT> <INPUT TYPE=RESET>
</FORM>
```



In the example above, the <P> and elements have been used to lay out the text and input fields. The HTML user agent is responsible for handling which field will currently get keyboard input.

Many platforms have existing conventions for forms, for example, using Tab and Shift keys to move the keyboard focus forwards and backwards between fields, and using the Enter key to submit the form. In the example, the SUBMIT and RESET buttons are specified explicitly with special purpose fields. The SUBMIT button is used to e-mail the form or send its contents to the server as specified by the ACTION attribute, while RESET resets the fields to their initial values. When the form consists of a single text field, it may be appropriate to leave such buttons out and rely on the Enter key.

The Input element is used for a large variety of types of input fields. To let users enter more than one line of text, use the Textarea element.

HTTP File Upload: It is possible to write forms that ask for files as input, rather than input boxes and

other simple elements such as checkboxes and radio buttons.

An example of such a form would be:

```
<FORM ENCTYPE="multipart/form-data" ACTION="_URL_" METHOD=POST>  
Send this file: <INPUT NAME="userfile" TYPE="file">  
<INPUT TYPE="submit" VALUE="Send File">  
</FORM>
```

NOTE : This method of file upload is **Netscape specific** and is essentially adoption of another IETF Internet Draft by the Netscape authors. The Internet Draft in question, "Form based file upload in HTML", suggested adding a `FILE` option to the `TYPE` attribute of the `INPUT` element, allowing an `ACCEPT` attribute for the `INPUT` element (which is a list of media types or type patterns allowed for the input) and allowing the `ENCTYPE` of a form to be `multipart/form-data`.

Such additions to the `<FORM>` element could prove invaluable for example, for companies providing technical support, or service providers, requesting data files.

Sample Questionnaire

Please fill out this questionnaire:

Your name:

Male

Female

Number in family:

Cities in which you maintain a residence:

◆ Kent

◆ Miami

◆ Other

Nickname:

Thank you for responding to this questionnaire.

<FORM> ... </FORM>

[Attributes](#)

[See Also](#)

The Form element is used to delimit a data input form. There can be several forms in a single document, but the Form element can't be nested.

The **ACTION** attribute is a URL specifying the location to which the contents of the form is submitted to elicit a response. If the **ACTION** attribute is missing, the URL of the document itself is assumed. The way data is submitted varies with the access protocol of the URL, and with the values of the **METHOD** and **ENCTYPE** attributes.

In general:

- the **METHOD** attribute selects variations in the protocol.
- the **ENCTYPE** attribute specifies the format of the submitted data in case the protocol does not impose a format itself.

The Level 2 specification defines and requires support for the HTTP access protocol only.

When the **ACTION** attribute is set to an HTTP URL, the **METHOD** attribute must be set to an HTTP method as defined by the HTTP method specification in the IETF draft HTTP standard. The default **METHOD** is **GET**, although for many applications, the **POST** method may be preferred. With the post method, the **ENCTYPE** attribute is a MIME type specifying the format of the posted data; by default, is `application/x-www-form-urlencoded`.

Under any protocol, the submitted contents of the form logically consist of name/value pairs. The names are usually equal to the **NAME** attributes of the various interactive elements in the form.

NOTE: The names are not guaranteed to be unique keys, nor are the names of form elements required to be distinct. The values encode the user's input to the corresponding interactive elements. Elements capable of displaying a textual or numerical value will return a name/value pair even when they receive no explicit user input.

The following are all attributes
of the <FORM> element.

ACTION

METHOD

ENCTYPE

All are detailed on this page.

Other <FORM> elements :

<INPUT> Element

<OPTION> Element

<SELECT> Element

<TEXTAREA> Element

Forms - <INPUT>

[Attributes](#)

[See Also](#)

The Input element represents a field whose contents may be edited by the user.

Attributes of the Input element:

ALIGN

Vertical alignment of the image. For use only with `TYPE=IMAGE` in HTML level 2. The possible values are exactly the same as for the `ALIGN` attribute of the image element.

CHECKED

Indicates that a checkbox or radio button is selected. Unselected checkboxes and radio buttons do not return name/value pairs when the form is submitted.

MAXLENGTH

Indicates the maximum number of characters that can be entered into a text field. This can be greater than specified by the `SIZE` attribute, in which case the field will scroll appropriately. The default number of characters is unlimited.

NAME

Symbolic name used when transferring the form's contents. The `NAME` attribute is required for most input types and is normally used to provide a unique identifier for a field, or for a logically related group of fields.

SIZE

Specifies the size or precision of the field according to its type. For example, to specify a field with a visible width of 24 characters:

```
e.g. : INPUT TYPE=text SIZE="24"
```

SRC

A URL or URN specifying an image. For use only with `TYPE=IMAGE` in HTML Level 2.

TYPE

Defines the type of data the field accepts. Defaults to free text. Several types of fields can be defined with the `type` attribute:

CHECKBOX : Used for simple Boolean attributes, or for attributes that can take multiple values at the same time. The latter is represented by a number of checkbox fields each of which has the same name. Each selected checkbox generates a separate name/value pair in the submitted data, even if this results in duplicate names. The default value for checkboxes is "on".

HIDDEN : No field is presented to the user, but the content of the field is sent with the submitted form. This value may be used to transmit state information about client/server interaction.

IMAGE : An image field upon which you can click with a pointing device, causing the form to be immediately submitted. The co-ordinates of the selected point are measured in pixel units from the upper-left corner of the image, and are returned (along with the other contents of the form) in two name/value pairs. The x-co-ordinate is submitted under the name of the field with `.x` appended, and the y- co-ordinate is submitted under the name of the field with `.y` appended. Any `VALUE` attribute is ignored. The image itself is specified by the `SRC` attribute, exactly as for the Image element.

NOTE: In a future version of the HTML specification, the `IMAGE` functionality may be folded into an enhanced `SUBMIT` field.

PASSWORD is the same as the `TEXT` attribute, except that text is not displayed as it is entered.

RADIO is used for attributes that accept a single value from a set of alternatives. Each radio button field in the group should be given the same name. Only the selected radio button in the group generates a name/value pair in the submitted data. Radio buttons require an explicit `VALUE` attribute.

RESET is a button that when pressed resets the form's fields to their specified initial values. The label to be displayed on the button may be specified just as for the `SUBMIT` button.

SUBMIT is a button that when pressed submits the form. You can use the `VALUE` attribute to provide a non-editable label to be displayed on the button. The default label is application-specific. If a `SUBMIT` button is pressed in order to submit the form, and that button has a `NAME` attribute specified, then that button contributes a name/value pair to the submitted data. Otherwise, a `SUBMIT` button makes no contribution to the submitted data.

TEXT is used for a single line text entry fields. Use in conjunction with the `SIZE` and `MAXLENGTH` attributes. Use the Textarea element for text fields which can accept multiple lines.

TEXTAREA is used for multiple-line text-entry fields. Use in conjunction with the `SIZE` and `MAXLENGTH` attributes.

VALUE

The initial displayed value of the field, if it displays a textual or numerical value; or the value to be returned when the field is selected, if it displays a Boolean value. This attribute is required for radio buttons.

The following are all allowed within
the <INPUT> element.

ALIGN

CHECKED

MAXLENGTH

NAME

SIZE

SRC

TYPE :

CHECKBOX; HIDDEN; IMAGE;

PASSWORD; RADIO; RESET;

SUBMIT; TEXT; TEXTAREA

VALUE

Other <FORM> elements :

<FORM> Element

<OPTION> Element

<SELECT> Element

<TEXTAREA> Element

Forms - <OPTION>

[Attributes](#)

[See Also](#)

The Option element can only occur within a Select element. It represents one choice, and can take these attributes:

DISABLED

Proposed.

SELECTED

Indicates that this option is initially selected.

VALUE

When present indicates the value to be returned if this option is chosen. The returned value defaults to the contents of the Option element.

The contents of the Option element is presented to the user to represent the option. It is used as a returned value if the `VALUE` attribute is not present.

The following are all allowed
within the <OPTION> element

DISABLED

SELECTED

VALUE

All are detailed on this page .

Other <FORM> elements :

<FORM> Element

<INPUT> Element

<SELECT> Element

<TEXTAREA> Element

Forms - <SELECT ...> ... </SELECT>

[Attributes](#)

[See Also](#)

The Select element allows the user to choose one of a set of alternatives described by textual labels. Every alternative is represented by the Option element.

Attributes are:

ERROR

Proposed.

MULTIPLE

The `MULTIPLE` attribute is needed when users are allowed to make several selections, e.g. `<SELECT MULTIPLE>`.

NAME

Specifies the name that will be submitted as a name/value pair.

SIZE

Specifies the number of visible items. If this is greater than one, then the resulting form control will be a list.

The `SELECT` element is typically rendered as a pull down or pop-up list. For example:

```
<SELECT NAME="flavor">
<OPTION>Vanilla
<OPTION>Strawberry
<OPTION>Rum and Raisin
<OPTION>Peach and Orange
</SELECT>
```

If no option is initially marked as selected, then the first item listed is selected.



| | |
|------------------|---|
| Strawberry | ↓ |
| Vanilla | |
| Strawberry | |
| Rum and Raisin | |
| Peach and Orange | |

The following are all allowed within
the `<SELECT>` element

ERROR
MULTIPLE
NAME
SIZE

All are detailed on this page.

Other <FORM> elements :

<FORM> Element

<INPUT> Element

<OPTION> Element

<TEXTAREA> Element

Forms - <TEXTAREA> ... </TEXTAREA>

[Attributes](#)

[See Also](#)

The `TEXTAREA` element lets users enter more than one line of text. For example:

```
e.g. : <TEXTAREA NAME="address" ROWS=64 COLS=6>
      HaL Computer Systems
      1315 Dell Avenue
      Campbell, California 95008
      </TEXTAREA>
```

The text up to the end element (`</TEXTAREA>`) is used to initialise the field's value. This end element is always required even if the field is initially blank. When submitting a form, lines in a `TEXTAREA` should be terminated using CR/LF.

In a typical rendering, the **ROWS** and **COLS** attributes determine the visible dimension of the field in characters. The field is rendered in a fixed-width font. HTML user agents should allow text to extend beyond these limits by scrolling as needed.

NOTE: In the initial design for forms, multi-line text fields were supported by the `Input` element with `TYPE=TEXT`. Unfortunately, this causes problems for fields with long text values. SGML's default (Reference Quantity Set) limits the length of attribute literals to only 240 characters. The HTML 2.0 SGML declaration increases the limit to 1024 characters.

Recent versions of the Netscape Navigator (from version 2.0) have introduced the **WRAP** attribute in the `TEXTAREA` element: Now it is possible to specify how to handle word-wrapping in text input areas in forms.

```
<TEXTAREA WRAP=OFF> -- the default setting - Wrapping doesn't happen. Lines are sent exactly as typed.
```

```
<TEXTAREA WRAP=VIRTUAL> -- The display word-wraps, but long lines are sent as one line without new-lines.
```

```
<TEXTAREA WRAP=PHYSICAL> -- The display word-wraps, and the text is transmitted at all wrap points.
```

NOTE : Word wrapping in a `TEXTAREA` text box is **Netscape specific**.

The following are all allowed within
the <TEXTAREA> element

[NAME](#)

[ROWS](#)

[COLS](#)

[WRAP](#)

All are detailed on this page.

Other <FORM> elements :

<FORM> Element

<INPUT> Element

<OPTION> Element

<SELECT> Element

Character Data

The characters between HTML elements represent text. A HTML document (including elements and text) is encoded using the coded character set specified by the "charset" parameter of the "text/html" media type. For levels defined in this specification, the "charset" parameter is restricted to "US-ASCII" or "ISO-8859-1". ISO-8859-1 encodes a set of characters known as Latin Alphabet No. 1, or simply Latin-1. Latin-1 includes characters from most Western European languages, as well as a number of control characters. Latin-1 also includes a non-breaking space, a soft hyphen indicator, 93 graphical characters, 8 unassigned characters, and 25 control characters.

Because non-breaking space and soft hyphen indicator are not recognised and interpreted by all HTML user agents, their use is discouraged.

There are 58 character positions occupied by control characters. See [Control Characters](#) for details on the interpretation of control characters.

Because certain special characters are subject to interpretation and special processing, information providers and HTML user agent implementors should follow the guidelines in the [Special Characters](#) section.

In addition, HTML provides [character entity references](#) and [numerical character references](#) to facilitate the entry and interpretation of characters by name and by numerical position.

Because certain characters will be interpreted as markup, they must be represented by entity references as described in [character](#) and/or [numerical](#) references.

Special Characters

Certain characters have special meaning in HTML documents. There are two printing characters which may be interpreted by an HTML application to have an effect of the format of the text:

Space

- Interpreted as a word space (place where a line can be broken) in all contexts except the Preformatted Text element.
- Interpreted as a nonbreaking space within the Preformatted Text element.

Hyphen

- Interpreted as a hyphen glyph in all contexts
- Interpreted as a potential word space by hyphenation engine

The following entity names are used in HTML, always prefixed by ampersand (&) and followed by a semicolon. They represent particular graphic characters which have special meanings in places in the markup, or may not be part of the character set available to the writer.

The following table lists each of the supported characters specified in the Numeric and Special Graphic entity set, along with its name, syntax for use, and description. This list is derived from ISO Standard 8879:1986//ENTITIES Numeric and Special Graphic//EN however HTML does not provide support for the entire entity set. Only the entities listed below are supported.

Glyph	Name	Syntax	Description
<	lt	<	Less than sign
>	gt	>	Greater than sign
&	amp	&	Ampersand
"	quot	"	Double quote sign

Control Characters

Control characters are non-printable characters that are typically used for communication and device control, as format effectors, and as information separators.

In SGML applications, the use of control characters is limited in order to maximise the chance of successful interchange over heterogenous networks and operating systems. In HTML, only three control characters are used: Horizontal Tab (HT, encoded as 9 decimal in US-ASCII and ISO-8859-1), Carriage Return, and Line Feed.

Horizontal Tab is interpreted as a word space in all contexts except preformatted text. Within preformatted text, the tab should be interpreted to shift the horizontal column position to the next position which is a multiple of 8 on the same line; that is, $col := ((col+8) \text{ div } 8) * 8$ (where div is integer division).

Carriage Return and Line Feed are conventionally used to represent end of line. For Internet Media Types defined as "text/*", the sequence CR/LF is used to represent an end of line. In practice, text/html documents are frequently represented and transmitted using an end of line convention that depends on the conventions of the source of the document; frequently, that representation consists of CR only, LF only, or CR/LF combination. In HTML, end of line in any of its variations is interpreted as a word space in all contexts except preformatted text. Within preformatted text, HTML interpreting agents should expect to treat any of the three common representations of end-of-line as starting a new line.

Numeric Character References

[See Also](#)

In addition to any mechanism by which characters may be represented by the encoding of the HTML document, it is possible to explicitly reference the printing characters of the ISO-8859-1 character encoding using a numeric character reference.

Two reasons for using a numeric character reference:

- the keyboard does not provide a key for the character, such as on U.S. keyboards which do not provide European characters
- the character may be interpreted as SGML coding, such as the ampersand (&), double quotes ("), the lesser (<) and greater (>) characters

Numeric character references are represented in an HTML document as SGML entities whose name is number sign (#) followed by a numeral from 32-126 and 161-255. The HTML DTD includes a numeric character for each of the printing characters of the ISO-8859-1 encoding, so that one may reference them by number if it is inconvenient to enter them directly:

the ampersand (&), double quotes ("), lesser (<) and greater (>) characters

The following entity names are used in HTML, always prefixed by ampersand (&) and followed by a semicolon. This list, sorted numerically, is derived from ISO-8859-1 8-bit single-byte coded graphic character set:

Reference	Description
�- 	Unused
		Horizontal tab

	Line feed
 - 	Unused
 	Space
!	Exclamation mark
"	Quotation mark
#	Number sign
$	Dollar sign
%	Percent sign
&	Ampersand
'	Apostrophe
(Left parenthesis
)	Right parenthesis
*	Asterisk
+	Plus sign
,	Comma
-	Hyphen
.	Period (fullstop)
/	Solidus (slash)
0- 9	Digits 0-9
:	Colon
;	Semi-colon
<	Less than
=	Equals sign
>	Greater than
?	Question mark
@	Commercial at

[Left square bracket
\	Reverse solidus (backslash)
]	Right square bracket
^	Caret
_	Horizontal bar
`	Acute accent
a- z	Letters a-z
{	Left curly brace
|	Vertical bar
}	Right curly brace
~	Tilde
 - 	Unused
¡	Inverted exclamation
¢	Cent sign
£	Pound sterling
¤	General currency sign
¥	Yen sign
¦	Broken vertical bar
§	Section sign
¨	Umlaut (dieresis)
©	Copyright
ª	Feminine ordinal
«	Left angle quote, guillemot left
¬	Not sign
­	Soft hyphen
®	Registered trademark
¯	Macron accent
°	Degree sign
±	Plus or minus
²	Superscript two
³	Superscript three
´	Acute accent
µ	Micro sign
¶	Paragraph sign
·	Middle dot
¸	Cedilla
¹	Superscript one
º	Masculine ordinal
»	Right angle quote, guillemot right
¼	Fraction one-fourth
½	Fraction one-half
¾	Fraction three-fourths
¿	Inverted question mark
À	Capital A, acute accent
Á	Capital A, grave accent
Â	Capital A, circumflex accent
Ã	Capital A, tilde
Ä	Capital A, dieresis or umlaut mark
Å	Capital A, ring
Æ	Capital AE diphthong (ligature)
Ç	Capital C, cedilla
È	Capital E, acute accent
É	Capital E, grave accent
Ê	Capital E, circumflex accent
Ë	Capital E, dieresis or umlaut mark
Ì	Capital I, acute accent

Í	Capital I, grave accent
Î	Capital I, circumflex accent
Ï	Capital I, dieresis or umlaut mark
Ð	Capital Eth, Icelandic
Ñ	Capital N, tilde
Ò	Capital O, acute accent
Ó	Capital O, grave accent
Ô	Capital O, circumflex accent
Õ	Capital O, tilde
Ö	Capital O, dieresis or umlaut mark
×	Multiply sign
Ø	Capital O, slash
Ù	Capital U, acute accent
Ú	Capital U, grave accent
Û	Capital U, circumflex accent
Ü	Capital U, dieresis or umlaut mark
Ý	Capital Y, acute accent
Þ	Capital THORN, Icelandic
ß	Small sharp s, German (sz ligature)
à	Small a, acute accent
á	Small a, grave accent
â	Small a, circumflex accent
ã	Small a, tilde
ä	Small a, dieresis or umlaut mark
å	Small a, ring
æ	Small ae diphthong (ligature)
ç	Small c, cedilla
è	Small e, acute accent
é	Small e, grave accent
ê	Small e, circumflex accent
ë	Small e, dieresis or umlaut mark
ì	Small i, acute accent
í	Small i, grave accent
î	Small i, circumflex accent
ï	Small i, dieresis or umlaut mark
ð	Small eth, Icelandic
ñ	Small n, tilde
ò	Small o, acute accent
ó	Small o, grave accent
ô	Small o, circumflex accent
õ	Small o, tilde
ö	Small o, dieresis or umlaut mark
÷	Division sign
ø	Small o, slash
ù	Small u, acute accent
ú	Small u, grave accent
û	Small u, circumflex accent
ü	Small u, dieresis or umlaut mark
ý	Small y, acute accent
þ	Small thorn, Icelandic
ÿ	Small y, dieresis or umlaut mark

[Character Entity References](#)

Character Entity References

[See Also](#)

Many of the Latin alphabet No. 1 set of printing characters may be represented within the text of an HTML document by a character entity.

Two reasons for using a character entity:

- the keyboard does not provide a key for the character, such as on U.S. keyboards which do not provide European characters
- the character may be interpreted as SGML coding, such as the ampersand (&), double quotes ("), the lesser (<) and greater (>) characters

A character entity is represented in an HTML document as an SGML entity whose name is defined in the HTML DTD. The HTML DTD includes a character entity for each of the SGML markup characters and for each of the printing characters in the upper half of Latin-1, so that one may reference them by name if it is inconvenient to enter them directly:

the ampersand (&); double quotes ("); lesser (<); and greater (>) characters

e.g.: Kurt Gödel was a famous logician and mathematician.

NOTE: To ensure that a string of characters is not interpreted as markup, represent all occurrences of <, >, and & by character or entity references.

NOTE: There are SGML features, CDATA and RCDATA, to allow most <, >, and & characters to be entered without the use of entity or character references. Because these features tend to be used and implemented inconsistently, and because they require 8-bit characters to represent non-ASCII characters, they are not used in this version of the HTML DTD. An earlier HTML specification included an Example element whose syntax is not expressible in SGML. No markup was recognised inside of the Example element. While HTML user agents are encouraged to support this idiom, its use is deprecated.

The following entity names are used in HTML, always prefixed by ampersand (&) and followed by a semicolon. The following table lists each of the characters specified in the Added Latin 1 entity set, along with its name, syntax for use, and description. This list is derived from ISO Standard 8879:1986//ENTITIES Added Latin 1//EN. HTML supports the entire entity set.

Name	Syntax	Description
Aacute	Á	Capital A, acute accent
Agrave	À	Capital A, grave accent
Acirc	Â	Capital A, circumflex accent
Atilde	Ã	Capital A, tilde
Aring	Å	Capital A, ring
Auml	Ä	Capital A, dieresis or umlaut mark
AElig	Æ	Capital AE diphthong (ligature)
Ccedil	Ç	Capital C, cedilla
Eacute	É	Capital E, acute accent
Egrave	È	Capital E, grave accent
Ecirc	Ê	Capital E, circumflex accent
Euml	Ë	Capital E, dieresis or umlaut mark
Iacute	Í	Capital I, acute accent
Igrave	Ì	Capital I, grave accent

Icirc	Î	Capital I, circumflex accent
Iuml	Ï	Capital I, dieresis or umlaut mark
ETH	Ð	Capital Eth, Icelandic
Ntilde	Ñ	Capital N, tilde
Oacute	Ó	Capital O, acute accent
Ograve	Ò	Capital O, grave accent
Ocirc	Ô	Capital O, circumflex accent
Otilde	Õ	Capital O, tilde
Ouml	Ö	Capital O, dieresis or umlaut mark
Oslash	Ø	Capital O, slash
Uacute	Ú	Capital U, acute accent
Ugrave	Ù	Capital U, grave accent
Ucirc	Û	Capital U, circumflex accent
Uuml	Ü	Capital U, dieresis or umlaut mark;
Yacute	Ý	Capital Y, acute accent
THORN	Þ	Capital THORN, Icelandic
Szlig	ß	Small sharp s, German (sz ligature)

aacute	á	Small a, acute accent
agrave	à	Small a, grave accent
acirc	â	Small a, circumflex accent
atilde	ã	Small a, tilde
aring	å	Small a, ring
auml	ä	Small a, dieresis or umlaut mark
aelig	æ	Small ae diphthong (ligature)
ccedil	ç	Small c, cedilla
eacute	é	Small e, acute accent
egrave	è	Small e, grave accent
ecirc	ê	Small e, circumflex accent
euml	ë	Small e, dieresis or umlaut mark
iacute	í	Small i, acute accent
igrave	ì	Small i, grave accent
icirc	î	Small i, circumflex accent
iuml	ï	Small i, dieresis or umlaut mark
eth	ð	Small eth, Icelandic
ntilde	ñ	Small n, tilde
oacute	ó	Small o, acute accent
ograve	ò	Small o, grave accent
ocirc	ô	Small o, circumflex accent
otilde	õ	Small o, tilde
ouml	ö	Small o, dieresis or umlaut mark
oslash	ø	Small o, slash
uacute	ú	Small u, acute accent
ugrave	ù	Small u, grave accent
ucirc	û	Small u, circumflex accent
uuml	ü	Small u, dieresis or umlaut mark
yacute	ý	Small y, acute accent
thorn	þ	Small thorn, Icelandic
yuml	ÿ	Small y, dieresis or umlaut mark

reg	®	Registered TradeMark
copy	©	Copyright
trade	&trade	TradeMark
nbspc	 c	Non breaking space

NOTE : The last three character entities are only supported in recent versions of Mosaic and Netscape. They may not appear as planned in early versions of these, or different browsers.

Numerical Character References

Tables

At present, the table HTML elements are :

[<TABLE> ... </TABLE>](#) - The Table delimiter.
[<TR ...> ... </TR>](#) - Used to specify number of rows in a table.
[<TD ...> ... </TD>](#) - Specifies table data cells.
[<TH ...> ... </TH>](#) - Table Header cell.
[<CAPTION ...> ... </CAPTION>](#) - Specifies the table Caption.

If these details are too confusing, there is also a [graphical guide to Tables](#) giving some example tables.

Some considerations about Tables

Blank cells which contain no displayable elements are not given borders. If a bordered but empty cell is required, put either a blank line or a non-breaking space in the cell. (<td> </td> or <td>
</td>)

The proposed HTML 3.0 spec. allows you to abuse row and column spans to create tables whose cells must overlap. Until this specification is finalised, don't do this, it looks awful.

Eventually, you may create a cell containing an image and it will possibly be rendered with the image off-centre. The reason for this could be due to the HTML being written like :

```
<TD>
<IMG SRC="url">
</TD>
```

That extra whitespace inside the cell and around the image gets collapsed into single space characters. It is these spaces (whose baselines by default align with the bottom of the image) that make the cell look lopsided. Instead, use :1

```
<TD><IMG SRC="url"></TD>
```

Also, please consider the user. If they're not using a recent browser that supports table elements, then page design will be completely lost. At present, use tables with caution

<TABLE> ... </TABLE>

[Attributes](#)

[See Also](#)

This is the main wrapper for all the other table elements, and other table elements will be ignored if they aren't wrapped inside of a <TABLE> . . . </TABLE> element. By default tables have no borders, borders will be added if the `BORDER` attribute is specified. At the time of writing, the <TABLE> element has an implied line break both before and after it. This is expected to change, allowing as much control over placement of tables as is currently available for the placement of images. Aligning them to various positions in a line of text, as well as shifting them to the left or right margins and wrapping text around them.

The <TABLE> element has the following attributes :

BORDER

This attribute appears in the <TABLE> element. If present, borders are drawn around all table cells. If absent, there are no borders, but by default space is left for borders, so the same table with and without the `BORDER` attribute will have the same width.

`BORDER=<value>`

By allowing the `BORDER` attribute to take a value, the document author gains two things. First they gain the ability to emphasise some tables with respect to others, a table with a border of four containing a sub-table with a border of one looks much nicer than if they both share the same default border width. Second, by explicitly setting border to zero they regain that space originally reserved for borders between cells, allowing particularly compact tables.

`CELLSPACING=<value>`

This is a new attribute for the <TABLE> element. By default Netscape uses a cell spacing of two. For those fussy about the look of their tables, this gives them a little more control. Like it sounds, cell spacing is the amount of space inserted between individual cells in a table.

`CELLPADDING=<value>`

This is a new attribute for the <TABLE> element. By default Netscape uses a cell padding of one. Cell padding is the amount of space between the border of the cell and the contents of the cell. Setting a cell padding of zero on a table with borders might look bad because the edges of the text could touch the cell borders.

```
<TABLE BORDER=0 CELLSPACING=0 CELLPADDING=0>
```

gives the most compact table possible.

`WIDTH=<value_or_percent>`

When this attribute appears in the <TABLE> element it is used to describe the desired width of this table, either as an absolute width in pixels, or a percentage of document width. Ordinarily complex heuristics are applied to tables and their cells to attempt to present a pleasing looking table. Setting the <WIDTH> attribute overrides those heuristics and instead effort is put into fitting the table into the desired width as specified. In some cases it might be impossible to fit all the table cells at the specified width, in which case Netscape will try and get as close as possible.

When this attribute appears on either the <TH> or <TD> element it is used to describe the desired width of the cell, either as an absolute width in pixels, or a percentage of table width. Ordinarily complex heuristics are applied to table cells to attempt to present a pleasing looking table. Setting the <WIDTH> attribute overrides those heuristics for that cell and instead effort is put into fitting the cell into the desired width as specified. In some cases it might be impossible to fit all the table cells at the specified widths, in which case Netscape will try and get as close as possible.

ALIGN

Microsofts **Internet Explorer** (version 2.0) has added support for the `ALIGN` attribute to the `<TABLE>` element. Like that used for [floating images](#), it allows a table to be aligned to the **left** or **right** of the page, allowing text to flow around the table. Also, as with floating images, it is necessary to have knowledge of the `<BR CLEAR=...>` element, to be able to organise the text so as to minimise any unwanted clashing.

VALIGN

Microsofts **Internet Explorer** (version 2.0) has added a `VALIGN` attribute to the `<TABLE>` element. It specifies that the text can be **top**- or **bottom**-aligned. The default is centre-aligned.

BGCOLOR

Microsofts **Internet Explorer** (version 2.0) has also included use of the `BGCOLOR` attribute in the `TABLE` element. It allows the background colour of the table to be specified, using either the specified [colour names](#), or a rrggbb hex triplet.

BORDERCOLOR

Microsofts **Internet Explorer** (version 2.0) includes support for this attribute which sets the border colour of the table. Any of the pre-defined [colour names](#) can be used, as well as any colour defined by a rrggbb hex triplet. It is necessary for the `BORDER` attribute to be present in the main `<TABLE>` element for border colouring to work.

BORDERCOLORLIGHT

Microsofts **Internet Explorer** (version 2.0) allows use of the `BORDERCOLORLIGHT` attribute to set independently, the lighter colour to be displayed on a 3-dimensional `<TABLE>` border. It is the opposite of `BORDERCOLORDARK`. Any of the pre-defined [colour names](#) can be used, as well as any colour defined by a rrggbb hex triplet. It is necessary for the `BORDER` attribute to be present in the main `<TABLE>` element for border colouring to work.

BORDERCOLORDARK

Microsofts **Internet Explorer** (version 2.0) allows use of the `BORDERCOLORDARK` attribute to set independently, the darker colour to be displayed on a 3-dimensional `<TABLE>` border. It is the opposite of `BORDERCOLORLIGHT`. Any of the pre-defined [colour names](#) can be used, as well as any colour defined by a rrggbb hex triplet. It is necessary for the `BORDER` attribute to be present in the main `<TABLE>` element for border colouring to work.

NOTE : The `BGCOLOR`, `BORDERCOLOR`, `BORDERCOLORLIGHT` and `BORDERCOLORDARK` attributes can also be used in `<TH>`, `<TR>` and `<TD>` elements, with the colour defined in the last element over-riding those defined before. E.g. if a `<TD>` element contains a `BORDERCOLOR` attribute setting, the setting specified will be used instead of any colour settings that may have been specified in the `<TR>` element, which in turn over-rides any colour settings in the `<TABLE>` element.

The following are all allowed within
the <TABLE> element

BORDER

CELLSPACING

CELLPADDING

WIDTH

ALIGN

VALIGN

BGCOLOR

BORDERCOLOR

BORDERCOLORLIGHT

BORDERCOLORDARK

<TR> Specifies number of rows in a table.

<TD> Data cell element.

<TH> Header element.

<CAPTION> Caption element.

Graphical examples of Tables

Table - <TR ...> ... </TR>

[Attributes](#)

[See Also](#)

This stands for table row. The number of rows in a table is exactly specified by how many <TR> elements are contained within it, regardless of cells that may attempt to use the ROWSPAN attribute to span into non-specified rows. <TR> can have both the ALIGN and VALIGN attributes, which if specified become the default alignments for all cells in this row.

The <TR> element can have the following attributes.

ALIGN

This controls whether text inside the table cell(s) is aligned to the left side of the row, the right side of the cell, or centred within the cell. Values are **left**, **center**, and **right**.

VALIGN

This attribute controls whether text inside the table cell(s) is aligned to the top of the row, the bottom of the cell, or vertically centred within the cell. It can also specify that all the cells in the row should be vertically aligned to the same baseline. Values are **top**, **middle**, **bottom** and **baseline**.

BGCOLOR

Microsofts **Internet Explorer** (version 2.0) has also included use of the [BGCOLOR](#) attribute in the TR element. It allows the background colour of the row to be specified, using either the specified [colour names](#), or a rrggbb hex triplet.

BORDERCOLOR

Microsofts **Internet Explorer** (version 2.0) includes support for this attribute which sets the border colour of the row. Any of the pre-defined [colour names](#) can be used, as well as any colour defined by a rrggbb hex triplet. It is necessary for the [BORDER](#) attribute to be present in the main [<TABLE>](#) element for border colouring to work.

BORDERCOLORLIGHT

Microsofts **Internet Explorer** (version 2.0) allows use of the BORDERCOLORLIGHT attribute to set independently, the lighter colour to be displayed on a 3-dimensional <TABLE> border. It is the opposite of BORDERCOLORDARK. Any of the pre-defined [colour names](#) can be used, as well as any colour defined by a rrggbb hex triplet. It is necessary for the [BORDER](#) attribute to be present in the main <TABLE> element for border colouring to work.

BORDERCOLORDARK

Microsofts **Internet Explorer** (version 2.0) allows use of the BORDERCOLORDARK attribute to set independently, the darker colour to be displayed on a 3-dimensional <TABLE> border. It is the opposite of BORDERCOLORLIGHT. Any of the pre-defined [colour names](#) can be used, as well as any colour defined by a rrggbb hex triplet. It is necessary for the [BORDER](#) attribute to be present in the main <TABLE> element for border colouring to work.

NOTE : The [BGCOLOR](#), [BORDERCOLOR](#), [BORDERCOLORLIGHT](#) and [BORDERCOLORDARK](#) attributes can also be used in [<TABLE>](#), [<TH>](#), and [<TD>](#) elements, with the colour defined in the last element over-riding those defined before. E.g. if a [<TD>](#) element contains a [BORDERCOLOR](#) attribute setting, the setting specified will be used instead of any colour settings that may have been specified in the <TR> element, which in turn over-rides any colour settings in the [<TABLE>](#) element.

[<TABLE>](#) - The Table element.

[<TD>](#) Table data cell element.

[<TH>](#) Table Header element.

[<CAPTION>](#) Caption element.

[Graphical examples of Tables](#)

The following are all allowed within
the <TR> element

ALIGN

VALIGN

BGCOLOR

BORDERCOLOR

BORDERCOLORLIGHT

BORDERCOLORDARK

Table - <TD ...> ... </TD>

[Attributes](#)

[See Also](#)

This stands for table data, and specifies a standard table data cell. Table data cells must only appear within table rows. Each row need not have the same number of cells specified as short rows will be padded with blank cells on the right. A cell can contain any of the HTML elements normally present in the body of an HTML document. The default alignment of table data is `ALIGN=left` and `VALIGN=middle`. These alignments are overridden by any alignments specified in the containing `<TR>` element, and those alignments in turn are overridden by any `ALIGN` or `VALIGN` attributes explicitly specified on this cell. By default lines inside of table cells can be broken up to fit within the overall cell width. Specifying the `NOWRAP` attribute for a `<TD>` prevents line breaking for that cell.

`<TD ...> ... </TD>` can accept the following attributes.

ALIGN

This attribute controls whether text inside the table cell(s) is aligned to the left side of the cell, the right side of the cell, or centred within the cell. Values are **left**, **center**, and **right**.

VALIGN

The `VALIGN` attribute controls whether text inside the table cell(s) is aligned to the top of the cell, the bottom of the cell, or vertically centred within the cell. It can also specify that all the cells in the row should be vertically aligned to the same baseline. Values are **top**, **middle**, **bottom** and **baseline**.

NOWRAP

If this attribute appears in any table cell (`<TH>` or `<TD>`) it means the lines within this cell cannot be broken to fit the width of the cell. Be cautious in use of this attribute as it can result in excessively wide cells.

COLSPAN

This attribute can appear in any table cell (`<TH>` or `<TD>`) and it specifies how many columns of the table this cell should span. The default `COLSPAN` for any cell is 1.

ROWSPAN

This attribute can appear in any table cell (`<TH>` or `<TD>`) and it specifies how many rows of the table this cell should span. The default `ROWSPAN` for any cell is 1. A span that extends into rows that were never specified with a `<TR>` will be truncated.

BGCOLOR

Microsofts **Internet Explorer** (version 2.0) has also included use of the `BGCOLOR` attribute in the `TD` element. It allows the background colour of the data cell to be specified, using either the specified [colour names](#), or a `rrggbb` hex triplet.

BORDERCOLOR

Microsofts **Internet Explorer** (version 2.0) includes support for this attribute which sets the border colour of the data cell. Any of the pre-defined [colour names](#) can be used, as well as any colour defined by a `rrggbb` hex triplet. It is necessary for the `BORDER` attribute to be present in the main `<TABLE>` element for border colouring to work.

BORDERCOLORLIGHT

Microsofts **Internet Explorer** (version 2.0) allows use of the `BORDERCOLORLIGHT` attribute to set independently, the lighter colour to be displayed on a 3-dimensional `<TABLE>` border. It is the opposite of `BORDERCOLORDARK`. Any of the pre-defined [colour names](#) can be used, as well as any colour defined by a `rrggbb` hex triplet. It is necessary for the `BORDER` attribute to be present in the main `<TABLE>` element for border colouring to work.

BORDERCOLORDARK

Microsofts **Internet Explorer** (version 2.0) allows use of the `BORDERCOLORDARK` attribute to set independently, the darker colour to be displayed on a 3-dimensional `<TABLE>` border. It is the opposite of `BORDERCOLORLIGHT`. Any of the pre-defined [colour names](#) can be used, as well as any colour defined by a rrggbb hex triplet. It is necessary for the [BORDER](#) attribute to be present in the main `<TABLE>` element for border colouring to work.

NOTE : The `BGCOLOR`, `BORDERCOLOR`, `BORDERCOLORDARK` and `BORDERCOLORLIGHT` attributes can also be used in [<TABLE>](#), [<TH>](#), and [<TR>](#) elements, with the colour defined in the last element over-riding those defined before. E.g. if a `<TD>` element contains a `BORDERCOLOR` attribute setting, the setting specified will be used instead of any colour settings that may have been specified in the [<TR>](#) element, which in turn over-rides any colour settings in the [<TABLE>](#) element.

<TABLE> - The Table element.

<TR> Specifies number of rows in a table.

<TH> Table Header element.

<CAPTION> Caption element.

Graphical examples of Tables

The following are all allowed within
the <TD> element

ROWSPAN

COLSPAN

ALIGN

VALIGN

NOWRAP

BGCOLOR

BORDERCOLOR

BORDERCOLORLIGHT

BORDERCOLORDARK

Table - <TH ...> ... </TH>

[Attributes](#)

[See Also](#)

This stands for table header. Header cells are identical to data cells in all respects, with the exception that header cells are in a **bold** FONT, and have a default `ALIGN=center`.

`<TH ...> ... </TH>` can contain the following attributes

ALIGN

The `ALIGN` attribute controls whether text inside the table cell(s) is aligned to the left side of the cell, the right side of the cell, or centred within the cell. Values are **left**, **center**, and **right**.

VALIGN

This attribute controls whether text inside the table cell(s) is aligned to the top of the cell, the bottom of the cell, or vertically centred within the cell. It can also specify that all the cells in the row should be vertically aligned to the same baseline. Values are **top**, **middle**, **bottom** and **baseline**.

NOWRAP

This attribute specifies that the lines within this cell cannot be broken to fit the width of the cell. Be cautious in use of this attribute as it can result in excessively wide cells.

COLSPAN

The `COLSPAN` attribute specifies how many columns of the table this cell should span. The default `COLSPAN` for any cell is 1.

ROWSPAN

This attribute specifies how many rows of the table this cell should span. The default `ROWSPAN` for any cell is 1. A span that extends into rows that were never specified with a `<TR>` will be truncated.

BGCOLOR

Microsofts **Internet Explorer** (version 2.0) has also included use of the `BGCOLOR` attribute in the `TH` element. It allows the background colour of the heading cell to be specified, using either the specified [colour names](#), or a `rrgbb` hex triplet.

BORDERCOLOR

Microsofts **Internet Explorer** (version 2.0) includes support for this attribute which sets the border colour of the heading cell. Any of the pre-defined [colour names](#) can be used, as well as any colour defined by a `rrgbb` hex triplet. It is necessary for the `BORDER` attribute to be present in the main `<TABLE>` element for border colouring to work.

BORDERCOLORLIGHT

Microsofts **Internet Explorer** (version 2.0) allows use of the `BORDERCOLORLIGHT` attribute to set independently, the lighter colour to be displayed on a 3-dimensional `<TABLE>` border. It is the opposite of `BORDERCOLORDARK`. Any of the pre-defined [colour names](#) can be used, as well as any colour defined by a `rrgbb` hex triplet. It is necessary for the `BORDER` attribute to be present in the main `<TABLE>` element for border colouring to work.

BORDERCOLORDARK

Microsofts **Internet Explorer** (version 2.0) allows use of the `BORDERCOLORDARK` attribute to set independently, the darker colour to be displayed on a 3-dimensional `<TABLE>` border. It is the opposite of `BORDERCOLORLIGHT`. Any of the pre-defined [colour names](#) can be used, as well as any colour defined by a `rrgbb` hex triplet. It is necessary for the `BORDER` attribute to be present in the main `<TABLE>` element for border colouring to work.

NOTE : The `BGCOLOR`, `BORDERCOLOR`, `BORDERCOLORDARK` and `BORDERCOLORLIGHT` attributes can also be used in `<TABLE>`, `<TD>`, and `<TR>` elements, with the colour defined in the last element over-riding those defined before. E.g. if a `<TD>` element contains a `BORDERCOLOR` attribute setting, the setting specified will be used instead of any colour settings that may have been specified in the `<TR>` element, which in turn over-rides any colour settings in the `<TABLE>` element.

<TABLE> - The Table element.

<TR> Specifies number of rows in a table.

<TD> Table data cell element.

<CAPTION> Caption element.

Graphical examples of Tables

The following are all allowed within
the <TH> element

ROWSPAN

COLSPAN

ALIGN

VALIGN

NOWRAP

BGCOLOR

BORDERCOLOR

BORDERCOLORLIGHT

BORDERCOLORDARK

Table - <CAPTION ...> ... </CAPTION>

[See Also](#)

This represents the caption for a table. <CAPTION> elements should appear inside the <TABLE> but not inside table rows or cells. The caption accepts an alignment attribute that defaults to `ALIGN=top` but can be explicitly set to `ALIGN=bottom`. Like table cells, any document body HTML can appear in a caption. Captions are always horizontally centred with respect to the table, and they may have their lines broken to fit within the width of the table.

The <CAPTION> element can accept the following attributes.

ALIGN

The `ALIGN` attribute controls whether the caption appears above or below the table, and can have the values **top** or **bottom**, defaulting to **top**. Microsoft's **Internet Explorer** allows use of **left**, **right** and **center** for the alignment of the <CAPTION> element. This forces the caption text to be aligned flush-left, flush-right, or centred within the boundaries of the table.

VALIGN

Microsoft's **Internet Explorer** allows use of the `VALIGN` attribute inside the <CAPTION> element. It specifies whether the caption text should be displayed at the **top** or **bottom** of the table.

<TABLE> - The Table element.

<TR> Specifies number of rows in a table.

<TD> Table data cell element.

<TH> Table Header element.

[Graphical examples of Tables](#)

Graphical Examples of Tables

Basic Tables :

[A Basic 3x2 table](#)

[Two demonstrations of ROWSPAN](#)

[Demonstration of COLSPAN](#)

[Demonstration of HEADERS, <TH ...>](#)

[Demonstration of COLSPAN plus <TH ...>](#)

[Demonstration of multiple headers plus COLSPAN](#)

[Demonstration of side headers](#)

[Demonstration of side headers plus ROWSPAN](#)

[Sample table using all of the above](#)

[Clever uses of ROWSPAN/COLSPAN](#)

Adjusting margins and borders :

[A table without borders](#)

[A table with a BORDER of 10](#)

[CELLPADDING and CELLSPACING](#)

Alignment, Captions and Sub-tables :

[Demonstration of multiple lines in a table](#)

[ALIGN=left|right|center](#)

[VALIGN=top|bottom|middle](#)

[CAPTION=top|bottom](#)

[Nested tables](#)

Table Widths and placing :

[Different table widths](#)

[Centering a table](#)

[Table width and nested tables](#)

[HEIGHT](#)

Internet Explorer specific attributes for tables.

[Floating tables](#)

[Colouring tables](#)

[Border Colouring](#)

[Caption Alignment](#)

Basic 3x2 Table

The following will render a Basic table having three columns and two rows.

```
<TABLE BORDER>  
<TR>  
  <TD>A</TD> <TD>B</TD> <TD>C</TD>  
</TR>  
<TR>  
  <TD>D</TD> <TD>E</TD> <TD>F</TD>  
</TR>  
</TABLE>
```



A	B	C
D	E	F

Two demonstrations of ROWSPAN

The following will render a table where item 2 spans two rows.

```
<TABLE BORDER>
  <TR>
    <TD>Item 1</TD>
    <TD ROWSPAN=2>Item 2</TD>
    <TD>Item 3</TD>
  </TR>
  <TR>
    <TD>Item 4</TD> <TD>Item 5</TD>
  </TR>
</TABLE>
```



The following will render a table where item 1 spans two rows.

```
<TABLE BORDER>
  <TR>
    <TD ROWSPAN=2>Item 1</TD>
    <TD>Item 2</TD> <TD>Item 3</TD> <TD>Item 4</TD>
  </TR>
  <TR>
    <TD>Item 5</TD> <TD>Item 6</TD> <TD>Item 7</TD>
  </TR>
</TABLE>
```



Item 1	Item 2	Item 3
Item 4		Item 5

Item 1	Item 2	Item 3	Item 4
	Item 5	Item 6	Item 7

Demonstration of COLSPAN

The following will render a table where item 2 spans two columns

```
<TABLE BORDER>
  <TR>
    <TD>Item 1</TD>
    <TD COLSPAN=2>Item 2</TD>
  </TR>
  <TR>
    <TD>Item 3</TD> <TD>Item 4</TD> <TD>Item 5</TD>
  </TR>
</TABLE>
```



Item 1	Item 2	
Item 3	Item 4	Item 5

Demonstration of Headers <TH>

The following will render a table with headers

```
<TABLE BORDER>
  <TR>
    <TH>Head1</TH> <TH>Head2</TH> <TH>Head3</TH>
  </TR>
  <TR>
    <TD>A</TD> <TD>B</TD> <TD>C</TD>
  </TR>
  <TR>
    <TD>D</TD> <TD>E</TD> <TD>F</TD>
  </TR>
</TABLE>
```



Head1	Head2	Head3
A	B	C
D	E	F

Demonstration of COLSPAN and <TH>

The following will render a table with headers that span two columns.

```
<TABLE BORDER>
  <TR>
    <TH COLSPAN=2>Head1</TH>
    <TH COLSPAN=2>Head2</TH>
  </TR>
  <TR>
    <TD>A</TD> <TD>B</TD> <TD>C</TD> <TD>D</TD>
  </TR>
  <TR>
    <TD>E</TD> <TD>F</TD> <TD>G</TD> <TD>H</TD>
  </TR>
</TABLE>
```



Head1		Head2	
A	B	C	D
E	F	G	H

Demonstration of multiple headers and COLSPAN

The following will render a table with multiple headers, one set of which is spanning two columns

```
<TABLE BORDER>
  <TR>
    <TH COLSPAN=2>Head1</TH>
    <TH COLSPAN=2>Head2</TH>
  </TR>
  <TR>
    <TH>Head 3</TH> <TH>Head 4</TH>
    <TH>Head 5</TH> <TH>Head 6</TH>
  </TR>
  <TR>
    <TD>A</TD> <TD>B</TD> <TD>C</TD> <TD>D</TD>
  </TR>
  <TR>
    <TD>E</TD> <TD>F</TD> <TD>G</TD> <TD>H</TD>
  </TR>
</TABLE>
```



Head1		Head2	
Head 3	Head 4	Head 5	Head 6
A	B	C	D
E	F	G	H

Demonstration of Side Headers

The following will render a table with the headers at the side.

```
<TABLE BORDER>
  <TR><TH>Head1</TH>
    <TD>Item 1</TD> <TD>Item 2</TD> <TD>Item 3</TD></TR>
  <TR><TH>Head2</TH>
    <TD>Item 4</TD> <TD>Item 5</TD> <TD>Item 6</TD></TR>
  <TR><TH>Head3</TH>
    <TD>Item 7</TD> <TD>Item 8</TD> <TD>Item 9</TD></TR>
</TABLE>
```



Head1	Item 1	Item 2	Item 3
Head2	Item 4	Item 5	Item 6
Head3	Item 7	Item 8	Item 9

Demonstration of side headers with ROWSPAN

The following will render a table with side headers, one of which spans multiple rows.

```
<TABLE BORDER>
  <TR><TH ROWSPAN=2>Head1</TH>
    <TD>Item 1</TD> <TD>Item 2</TD> <TD>Item 3</TD> <TD>Item 4</TD>
  </TR>
  <TR>
    <TD>Item 5</TD> <TD>Item 6</TD> <TD>Item 7</TD> <TD>Item 8</TD>
  </TR>
  <TR><TH>Head2</TH>
    <TD>Item 9</TD> <TD>Item 10</TD> <TD>Item 3</TD> <TD>Item 11</TD>
  </TR>
</TABLE>
```



Head1	Item 1	Item 2	Item 3	Item 4
	Item 5	Item 6	Item 7	Item 8
Head2	Item 9	Item 10	Item 3	Item 11

Sample table using all of the above

The following will render a table using all of the above attributes.

```
<TABLE BORDER>
  <TR>
    <TD><TH ROWSPAN=2></TH>
    <TH COLSPAN=2>Average</TH></TD>
  </TR>
  <TR>
    <TD><TH>Height</TH><TH>Weight</TH></TD>
  </TR>
  <TR>
    <TH ROWSPAN=2>Gender</TH>
    <TH>Males</TH><TD>1.9</TD><TD>0.003</TD>
  </TR>
  <TR>
    <TH>Females</TH><TD>1.7</TD><TD>0.002</TD>
  </TR>
</TABLE>
```



		Average	
		Height	Weight
Gender	Males	1.9	0.003
	Females	1.7	0.002

Clever use of ROWSPAN/COLSPAN

The following will render a table that uses both ROWSPAN=2 and COLSPAN=2 on side and bottom cells.

```
<TABLE BORDER>
  <TR>
    <TD ALIGN=center ROWSPAN=2 COLSPAN=2>A</TD>
    <TD>1</TD>
    <TD>2</TD>
  </TR>
  <TR>
    <TD>3</TD>
    <TD>4</TD>
  </TR>
  <TR>
    <TD ALIGN=center ROWSPAN=2 COLSPAN=2>C</TD>
    <TD ALIGN=center ROWSPAN=2 COLSPAN=2>D</TD>
  </TR>
  <TR>
  </TR>
</TABLE>
```



A	1	2
	3	4
C	D	

A Table with no borders

The following will render a table with no borders.

```
<TABLE>
  <TR>
    <TD>Item 1</TD> <TD ROWSPAN=2>Item 2</TD> <TD>Item 3</TD>
  </TR>
  <TR>
    <TD>Item 4</TD> <TD>Item 5</TD>
  </TR>
</TABLE>
```



Item 1 Item 2 Item 3
Item 4 Item 5

A table with a border of 10

The following will render a table with a border of 10.

```
<TABLE BORDER=10>
  <TR>
    <TD>Item 1</TD> <TD> Item 2</TD>
  </TR>
  <TR>
    <TD>Item 3</TD> <TD>Item 4</TD>
  </TR>
</TABLE>
```



Item 1	Item 2
Item 3	Item 4

CELLPADDING and CELLSPACING

The following renders a table using `CELLPADDING`, but no `CELLSPACING`

```
<TABLE BORDER CELLPADDING=10 CELLSPACING=0>
<TR>
  <TD>A</TD> <TD>B</TD> <TD>C</TD>
</TR>
<TR>
  <TD>D</TD> <TD>E</TD> <TD>F</TD>
</TR>
</TABLE>
```



The following renders a table using `CELLSPACING` but no `CELLPADDING`

```
<TABLE BORDER CELLPADDING=0 CELLSPACING=10>
<TR>
  <TD>A</TD> <TD>B</TD> <TD>C</TD>
</TR>
<TR>
  <TD>D</TD> <TD>E</TD> <TD>F</TD>
</TR>
</TABLE>
```



The following renders a table using `CELLPADDING` and `CELLSPACING`

```
<TABLE BORDER CELLPADDING=10 CELLSPACING=10>
<TR>
  <TD>A</TD> <TD>B</TD> <TD>C</TD>
</TR>
<TR>
  <TD>D</TD> <TD>E</TD> <TD>F</TD>
</TR>
</TABLE>
```



The following renders a table using `CELLSPACING`, `CELLPADDING` and specifying a `BORDER`.

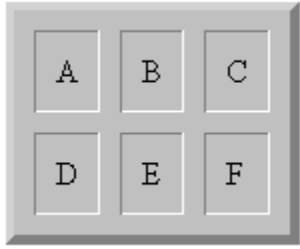
```
<TABLE BORDER=5 CELLPADDING=10 CELLSPACING=10>
<TR>
  <TD>A</TD> <TD>B</TD> <TD>C</TD>
</TR>
<TR>
  <TD>D</TD> <TD>E</TD> <TD>F</TD>
</TR>
</TABLE>
```



A	B	C
D	E	F

A	B	C
D	E	F

A	B	C
D	E	F



Multiple lines in a table

The following will render a table with multiple lines of text in cells.

```
<TABLE BORDER>
  <TR>
    <TH>January</TH>
    <TH>February</TH>
    <TH>March</TH>
  </TR>
  <TR>
    <TD>This is cell 1</TD>
    <TD>Cell 2</TD>
    <TD>Another cell,<br> cell 3</TD>
  </TR>
  <TR>
    <TD>Cell 4</TD>
    <TD>and now this<br>is cell 5</TD>
    <TD>Cell 6</TD>
  </TR>
</TABLE>
```



January	February	March
This is cell 1	Cell 2	Another cell, cell 3
Cell 4	and now this is cell 5	Cell 6

ALIGN=left|right|center

The following will render a table showing different possible alignments of text.

NOTE : this formatting can be applied to individual cells or whole rows.

```
<TABLE BORDER>
  <TR>
    <TH>January</TH>
    <TH>February</TH>
    <TH>March</TH>
  </TR>
  <TR ALIGN=center>
    <TD>all aligned center</TD>
    <TD>Cell 2</TD>
    <TD>Another cell,<br> cell 3</TD>
  </TR>
  <TR>
    <TD ALIGN=right>aligned right</TD>
    <TD ALIGN=center>aligned to center</TD>
    <TD>default,<br>aligned left</TD>
  </TR>
</TABLE>
```



January	February	March
all aligned center	Cell 2	Another cell, cell 3
aligned right	aligned to center	default, aligned left

VALIGN=top|bottom|middle

The following will render a table showing different possible vertical text alignments possible within table cells.

NOTE : this formatting can be applied to individual cells or whole rows.

```
<TABLE BORDER>
  <TR>
    <TH>January</TH>
    <TH>February</TH>
    <TH>March</TH>
  </TR>
  <TR VALIGN=top>
    <TD>all aligned to top</TD>
    <TD>and now this<br>is cell 2</TD>
    <TD>Cell 3</TD>
  </TR>
  <TR>
    <TD VALIGN=top>aligned to the top</TD>
    <TD VALIGN=bottom>aligned to the bottom</TD>
    <TD>default alignment,<br>center</TD>
  </TR>
</TABLE>
```



January	February	March
all aligned to top	and now this is cell 2	Cell 3
aligned to the top	aligned to the bottom	default alignment, center

CAPTION=top|bottom

The following will render a table with a caption at the top.

```
<TABLE BORDER>
  <CAPTION ALIGN=top>A top CAPTION</CAPTION>
  <TR>
    <TH>January</TH>
    <TH>February</TH>
    <TH>March</TH>
  </TR>
  <TR>
    <TD>This is cell 1</TD>
    <TD>Cell 2</TD>
    <TD>Another cell,<br> cell 3</TD>
  </TR>
</TABLE>
```



The following will render a table with a caption at the bottom

```
<TABLE BORDER>
  <CAPTION ALIGN=bottom>A bottom CAPTION</CAPTION>
  <TR>
    <TH>January</TH>
    <TH>February</TH>
    <TH>March</TH>
  </TR>
  <TR>
    <TD>This is cell 1</TD>
    <TD>Cell 2</TD>
    <TD>Another cell,<br> cell 3</TD>
  </TR>
</TABLE>
```



A top CAPTION

January	February	March
This is cell 1	Cell 2	Another cell, cell 3

January	February	March
This is cell 1	Cell 2	Another cell, cell 3

A bottom CAPTION

Nested Tables

The following will render a table within a table. Table ABCD is inside table 123456.

```
<TABLE BORDER>
  <TR> <!-- ROW 1, TABLE 1 -->
    <TD>1</TD>
    <TD>2</TD>
    <TD>3
      <TABLE BORDER>
        <TR> <!-- ROW 1, TABLE 2 -->
          <TD>A</TD>
          <TD>B</TD>
        </TR>
        <TR> <!-- ROW 2, TABLE 2 -->
          <TD>C</TD>
          <TD>D</TD>
        </TR>
      </TABLE>
    </TD>
  </TR>
  <TR> <!-- ROW 2, TABLE 1 -->
    <TD>4</TD>
    <TD>5</TD>
    <TD>6</TD>
  </TR>
</TABLE>
```



		3
1	2	A B
		C D
4	5	6

Setting table width

The following will render a table of width 50% (of page width).

```
<TABLE BORDER WIDTH="50%">
  <TR>
    <TD>Width=50%</TD><TD>Width=50%</TD>
  </TR>
  <TR>
    <TD>3</TD><TD>4</TD>
  </TR>
</TABLE>
```



Note that if the cells contain non-identical width data, it affects the cell width relative to the table width :

```
<TABLE BORDER WIDTH="50%">
  <TR>
    <TD>Item width affects cell size</TD><TD>2</TD>
  </TR>
  <TR>
    <TD>3</TD><TD>4</TD>
  </TR>
</TABLE>
```



NOTE : This table would appear aligned to the left of the page and half the width of the page

Width=50%	Width=50%
3	4

NOTE : This table would appear aligned to the left of the page and half the width of the page

Item width affects cell size	2
3	4

Centering a table

The following would render a table in the centre of the page.

```
<CENTER>  
<TABLE BORDER WIDTH="50%">  
  <TR>  
    <TD>A</TD> <TD>B</TD> <TD>C</TD>  
  </TR>  
  <TR>  
    <TD>D</TD> <TD>E</TD> <TD>F</TD>  
  </TR>  
</TABLE>  
</CENTER>
```



NOTE : This table would be rendered aligned in the centre of the viewing page.

A	B	C
D	E	F

Table width and nested tables

The following will render two nested tables, using table width attribute to specify the size for the secondary table

```
<TABLE BORDER WIDTH="50%">
  <TR>
    <TD>Item 1</TD><TD>Item 2</TD>
  </TR>
  <TR>
    <TD>
      <TABLE BORDER WIDTH=100%>
        <TR>
          <TD>Item A</TD><TD>Item B</TD>
        </TR>
      </TABLE>
    </TD>
    <TD>Item 4</TD>
  </TR>
</TABLE>
```



Item 1	Item 2
Item A	Item B
	Item 4

Table Height

The following will render a table of height 15% (relative to the viewing page)

```
<TABLE BORDER WIDTH="50%" HEIGHT="15%">
  <TR>
    <TD>HEIGHT=15%</TD> <TD>Item 2</TD>
  </TR>
  <TR>
    <TD>3</TD><TD>4</TD>
  </TR>
</TABLE>
```



NOTE : This table would be rendered at a height of 15% relative to the viewing page.

HEIGHT=15%	Item 2
3	4

Background Colour Chart
<BODY> Element

Floating tables

The following will render two small tables, both of which are floating, with text wrapping around the tables. **NOTE** : The screenshot was taken using Internet Explorer. Other browsers may not display the tables as they are shown here.

```
<TABLE ALIGN=left BORDER WIDTH=50%>
  <TR>
    <TD>This is a two row table</TD>
  </TR>
  <TR>
    <TD>It is aligned to the left of the page</TD>
  </TR>
</TABLE>
```

This text will be to the right of the table, and will fall neatly beside the table

```
<BR CLEAR=all>
<HR>
```

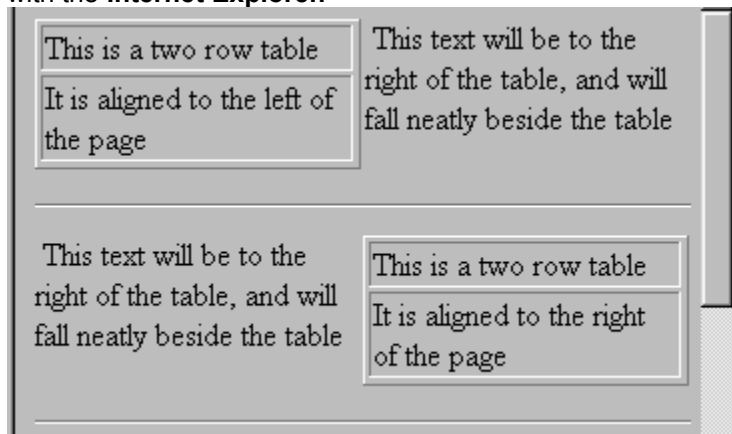
```
<TABLE ALIGN=right BORDER WIDTH=50%>
  <TR>
    <TD>This is a two row table</TD>
  </TR>
  <TR>
    <TD>It is aligned to the right of the page</TD>
  </TR>
</TABLE>
```

This text will be to the right of the table, and will fall neatly beside the table

```
<BR CLEAR=all>
<HR>
```



NOTE : Table alignment is currently only possible with the **Internet Explorer**.



Colouring Tables

The following table shows the use of the `BGCOLOR` and `BORDERCOLOR` attributes in the `<TABLE>`, `<TR>`, `<TD>` and `<TH>` elements. It shows the order of rendering as well. I.e. the `BORDERCOLOR` and `BGCOLOR` settings used in the `<TD>` elements over-ride those set in the `<TR>` elements, which in turn over-ride those set in the `<TABLE>` element.

```
<TABLE BORDER BGCOLOR=Silver BORDERCOLOR=Black WIDTH=50%>
  <TR>
    <TD>This is the first cell</TD>
    <TD>This is the second cell</TD>
  </TR>
  <TR BORDERCOLOR=Red BGCOLOR=Green>
    <TD>This is the third cell</TD>
    <TD>This is the fourth cell</TD>
  </TR>
  <TR BORDERCOLOR=Red BGCOLOR=Green>
    <TD BORDERCOLOR=Yellow>This is the fifth cell</TD>
    <TD BGCOLOR=White>This is the sixth cell</TD>
  </TR>
</TABLE>
```



NOTE : For the fifth cell, where a `BGCOLOR` is not specified, it adopts the background colour most recently specified, i.e. Green. The same is true for the sixth cell that adopts the most recent border colour of Red.

NOTE : For table cell and row border colouring to work, the `BORDER` attribute of the `<TABLE>` element must be present.

The following table was rendered in Microsofts Internet Explorer. Other browsers may not support colouring of tables.

This is the first cell	This is the second cell
This is the third cell	This is the fourth cell
This is the fifth cell	This is the sixth cell

The first and second cells have no colour settings, so adopt those set in the main <TABLE> element.

I.e. BORDERCOLOR=Black and BGCOLOR=Silver.

The third and fourth cells have BORDERCOLOR=Red and BGCOLOR=Green specified for their whole row, over-riding the settings in the main <TABLE> element.

The fifth and sixth cells have BORDERCOLOR=Yellow and BGCOLOR=White respectively, over-riding the settings in their parent <TR> and <TABLE> elements.

Border Colouring

The following table shows the use of the `BORDERCOLORLIGHT` and `BORDERCOLORDARK` attributes, specifying the respective light and dark colours to be used when rendering the 3-D border of a table.

NOTE : Use of these attributes is **Internet Explorer** specific.

```
<TABLE BORDER BORDERCOLORDARK=Red BORDERCOLORLIGHT=Yellow>
  <TR>
    <TD>Data cell</TD>
    <TD>Data cell</TD>
  </TR>
</TABLE>
```



NOTE : Border colouring is currently only possible with the **Internet Explorer**.

Data cell	Data cell
-----------	-----------

Caption Alignment

The following tables use various **Internet Explorer** specific `<CAPTION>` attributes.

This table renders with the caption at the top left of the table.

```
<TABLE BORDER>
<CAPTION VALIGN=top ALIGN=left>Table caption</CAPTION>
<TR>
  <TD>Cell no. 1</TD>
  <TD>Cell no. 2</TD>
</TR>
<TR>
  <TD>Cell no. 3</TD>
  <TD>Cell no. 4</TD>
</TR>
</TABLE>
```



This renders a table with the caption at the bottom right of the table.

```
<TABLE BORDER>
<CAPTION VALIGN=bottom ALIGN=right>Table caption</CAPTION>
<TR>
  <TD>Cell no. 1</TD>
  <TD>Cell no. 2</TD>
</TR>
<TR>
  <TD>Cell no. 3</TD>
  <TD>Cell no. 4</TD>
</TR>
</TABLE>
```



NOTE : This table are currently only possible with the **Internet Explorer**.

Table caption

Cell no. 1	Cell no. 2
Cell no. 3	Cell no. 4

NOTE : This table is currently only possible with the **Internet Explorer**.

Cell no. 1	Cell no. 2
Cell no. 3	Cell no. 4

Table caption

Dynamic Documents

[See Also](#)

Recent browsers (i.e. Netscape and the Internet Explorer) support a couple of different mechanisms for dynamic documents. These are documents that are updated on a periodic, or frequent basis)

These mechanisms are called "server push" and "client pull", and are based on existing standards (including the standard MIME multipart mechanism and the HTML 2.0 `META` element).

[Server push](#)

The server sends down a chunk of data; the browser display the data but leaves the connection open; whenever the server wants it sends more data and the browser displays it, leaving the connection open; at some later time the server sends down yet more data and the browser displays it; etc.

[Client pull](#)

The server sends down a chunk of data, including a directive (in the HTTP response or the document header) that says "reload this data in 5 seconds", or "go load this other URL in 10 seconds". After the specified amount of time has elapsed, the client does what it was told -- either reloading the current data or getting new data.

[JavaScript](#)

Netscape Navigator 2.0 and Netscape Navigator Gold 2.0 provide flexible, lightweight programmability via the Netscape scripting language, a programmable API that allows cross-platform scripting of events, objects, and actions. It allows the page designer to access events such as startups, exits, and user mouse clicks. Based on the Java language, the Netscape scripting language extends the programmatic capabilities of Netscape Navigator to a wide range of authors and is easy enough for anyone who can compose HTML. Use the Netscape scripting language to glue HTML, inline plug-ins, and Java Applets to each other.

In server push, a HTTP connection is held open for an indefinite period of time (until the server knows it is done sending data to the client and sends a terminator, or until the client interrupts the connection). In client pull, HTTP connections are never held open; rather, the client is told when to open a new connection, and what data to fetch when it does so.

In server push, the magic is accomplished by using a variant of the MIME message format "multipart/mixed", which lets a single message (or HTTP response) contain many data items. In client pull, the magic is accomplished by an HTTP response header (or equivalent HTML element) that tells the client what to do after some specified time delay.

Server Push/Client Pull considerations.

Server push is generally more efficient than client pull, since a new connection doesn't need to be opened for each new piece of data. Since a connection is held open over time, even when no data is being transferred, the server must be willing to accept dedicated allocation of a TCP/IP port, which may be an issue for servers with a sharply limited number of TCP/IP ports.

Client pull is generally less efficient, since a new connection must be opened for each new piece of data. However, no connection is held open over time.

Note that in real world situations it is common for establishment of a new connection to take a significant amount of time -- i.e., one second or more. Given that this is the case, server push will probably be generally preferable for end-user performance reasons, particularly for information that is frequently updated.

Another consideration is that the server has comparatively more control in the server push situation than in the client pull situation. One example: there is one distinct open connection for each instance of server push in use, and the server can elect to (for example) shut down such a connection at any time

(e.g., via a cron daemon) without requiring a whole lot of logic in the server. On the other hand, the same application using client pull will look like many independent connections to the server, and the server may need to have a considerable level of complexity in order to manage the situation (e.g., associating client pull requests with particular end users to figure out who to stop sending new "Refresh" headers to).

An Important Note On Server Push And Shell Scripts: If a CGI program is written as a shell script, and the script implements some form of server push where the connection is expected to be open for a long time (e.g. an infinitely long stream of images representing live video), then the shell script normally will not notice when/if the user severs the connection on the client side (e.g., by pressing the "Stop" button) and will continue running. This is bad, as server resources will be thereafter consumed wastefully and uselessly. The easiest way to work around this shell script limitation is to implement such CGI programs using a language like Perl or C -- such programs will terminate properly when the user breaks the connection.

[Server Push](#)
[Client Pull](#)
[<META> Element](#)
[JavaScript](#)

Dynamic Documents - Server Push

[See Also](#)

Server push is the other dynamic document mechanism, complementing [client pull](#).

In contrast to client pull, server push takes advantage of a connection that's held open over multiple responses, so the server can send down more data any time it wants. The obvious major advantage is that the server has total control over when and how often new data is sent down. Also, this method can be more efficient, since new HTTP connections don't have to be opened all the time. The downside is that the open connection consumes a resource on the server side while it's open (only when the server knows it wants this to happen, though). Also, server push has two other advantages: one is that a server push is easily interruptible (you can just hit "Stop" and interrupt the connection). The other advantage will be discussed later.

First, a short review: the MIME message format is used by HTTP to encapsulate data returned from a server in response to a request. Typically, an HTTP response consists of only a single piece of data. However, MIME has a standard facility for representing many pieces of data in a single message (or HTTP response). This facility uses a standard MIME type called "multipart/mixed"; a multipart/mixed message looks something like:

```
Content-type: multipart/mixed;boundary=ThisRandomString

--ThisRandomString
Content-type: text/plain

Data for the first object.

--ThisRandomString
Content-type: text/plain

Data for the second and last object.

--ThisRandomString--
```

The above message contains two data blocks, both of type "text/plain". The final two dashes after the last occurrence of "ThisRandomString" indicate that the message is over; there is no more data.

For server push we use a variant of "multipart/mixed" called "multipart/x-mixed-replace". The "x-" indicates this type is experimental. The "replace" indicates that each new data block will cause the previous data block to be replaced -- that is, new data will be displayed instead of (not in addition to) old data.

Here's an example of "multipart/x-mixed-replace" in action:

```
Content-type: multipart/x-mixed-replace;boundary=ThisRandomString

--ThisRandomString
Content-type: text/plain

Data for the first object.

--ThisRandomString
Content-type: text/plain

Data for the second and last object.

--ThisRandomString--
```

The key to the use of this technique is that the server does not push the whole "multipart/x-mixed-replace" message down all at once but rather sends down each successive data block whenever it sees fit. The HTTP connection stays open all the time, and the server pushes down new data blocks as rapidly or as infrequently as it wants, and in between data blocks the browser simply sits and waits for more data in the current window. The user can even go off and do other things in other windows; when the server has more data to send, it just pushes another data block down the pipe, and the appropriate window updates itself.

Here's exactly what happens:

Following in the tradition of the standard "multipart/mixed", "multipart/x-mixed-replace" messages are composed using a unique boundary line that separates each data object. Each data object has its own headers, allowing for an object-specific content type and other information to be specified.

The specific behaviour of "multipart/x-mixed-replace" is that each new data object replaces the previous data object. The browser gets rid of the first data object and instead displays the second data object.

A "multipart/x-mixed-replace" message doesn't have to end! That is, the server can just keep the connection open forever and send down as many new data objects as it wants. The process will then terminate if the user is no longer displaying that data stream in a browser window or if the browser severs the connection (e.g. the user presses the "Stop" button). We expect this will be the typical way people will use server push.

The previous document will be cleared and the browser will begin displaying the next document when the "Content-type" header is found, or at the end of the headers otherwise, for a new data block.

The current data block (document) is considered finished when the next message boundary is found.

Together, the above two items mean that the server should push down the pipe: a set of headers (most likely including "Content-type"), the data itself, and a separator (message boundary). When the browser sees the separator, it knows to sit still and wait indefinitely for the next data block to arrive.

Putting it all together, here's a Unix shell script that will cause the browser to display a new listing of processes running on a server every 5 seconds:

```
#!/bin/sh
echo "HTTP/1.0 200"
echo "Content-type: multipart/x-mixed-replace;boundary=---ThisRandomString---"
echo ""
echo "---ThisRandomString---"
while true
do
echo "Content-type: text/html"
echo ""
echo "h2Processes on this machine updated every 5 seconds/h2"
echo "time: "
date
echo "p"
echo "plaintext"
ps -el
echo "---ThisRandomString---"
sleep 5
done
```

Note that the boundary is sent to the browser before the sleep statement. This ensures that the browser will flush its buffers and display all the data that's been received up to that point to the user.

NCSA HTTPD users must not use any spaces in the content type, this includes the boundary argument. NCSA HTTPD will only accept a single string with no white space as a content type. If any spaces are in the line (besides the one right after the colon) any text after the white space will be truncated.

As an example, the following will work:

```
Content-type: multipart/x-mixed-replace;boundary=ThisRandomString
```

The following will not work:

```
Content-type: multipart/x-mixed-replace; boundary=ThisRandomString
```

The other advantage of server push is that it can be used for individual inlined images! A document that contains an image can be made to update by the server on a regular basis or at any time the server wants. Just have the `SRC` attribute of the `IMG` element point to an URL for which the server pushes a series of images.

If server push is used for an individual inlined image, the image will get replaced inside the document each time a new image is pushed -- the document itself won't get touched (assuming it isn't separately subject to server push) -- poor man's animation inlined into a static document.

Client Pull
JavaScript

Dynamic Documents - Client Pull

[See Also](#)

A simple use of client pull is to cause a document to be automatically reloaded on a regular basis. For example, consider the following document :

```
<META HTTP-EQUIV="Refresh" CONTENT=1>
<TITLE>Document ONE</TITLE>

<H1>This is Document ONE!</H1>

Here's some text. <P>
```

If loaded into a browser supporting Dynamic Documents (Netscape 1.1 and above. Microsofts Internet Explorer and Mosaic also support client pull), it would re-load itself every second.

Simply put, the `META` element (a standard HTML 3.0 element, for simulating HTTP response headers in HTML documents) is telling the browser that it should pretend that the HTTP response when the document was loaded included the following header:

```
Refresh: 1
```

That HTTP header, in turn, tells the browser to reload (refresh) this document after 1 second has elapsed. If a 12 second delay was required, the following HTML directive would have been used:

```
META HTTP-EQUIV="Refresh" CONTENT=12
```

...which is equivalent to this HTTP response header:

```
Refresh: 12
```

NOTE: the `META` element should be used on the first line of a HTML document.

A couple of things to notice:

In this example, a new "Refresh" directive (via either the `META` element or the Refresh HTTP response header) is given as a part of every retrieval. This is an important point. Each individual "Refresh" directive is one-shot and non-repeating. The directive doesn't say "go get this page every 6 seconds from now until infinity"; it says "go get this page in 6 seconds".

If continuous reloading is required, the directive needs to be given on each retrieval. If the document only needs to be reloaded once, only give the directive on the first retrieval. Once given the directive, the browser will do the specified retrieval after the specified amount of time. The only way to cause it not to happen is to not have an open window that contains the document.

This also means that if an "infinite reload" situation is set up (as the example above does), the only way it can be interrupted is by pressing the "Back" button or otherwise going to a different URL in the current window (or, equivalently, by closing the current window).

So another thing to do, in addition to causing the current document to reload, is to cause another document to be reloaded in n seconds in place of the current document. This is easy. The HTTP response header will look like this:

```
Refresh: 12; URL=http://foo.bar/blatz.html
```

The corresponding `META` element would be:


```
<META HTTP-EQUIV="Refresh" CONTENT="12; URL=http://foo.bar/blatz.html">
```

Important note: the URL needs to be fully qualified (e.g. <http://whatever/whatever>). That is, don't use a relative URL.

Consider the following example documents, "doc2.html" and "doc3.html", each of which causes the other to load (so if one of them is loaded, the browser will flip back and forth between them indefinitely)

```
<META HTTP-EQUIV=REFRESH CONTENT="1; URL=http://machine/doc3.html">
<TITLE>Document TWO</TITLE>
```

```
<H1>This is Document TWO!</H1>
```

```
Here's some other text. <P>
```

```
<META HTTP-EQUIV=REFRESH CONTENT="1; URL=http://machine/doc2.html">
<TITLE>Document THREE</TITLE>
```

```
<H1>This is Document THREE!</H1>
```

```
Here's yet more text. <P>
```

On loading one of the documents; the browser will load the other in 1 second, then the first in another second, then the second again in another second, and so on forever.

How do you make it stop? The easiest way is to either close the window, or put a link in the document(s) that points to somewhere else. Remember, any retrieval of any document can cause the whole process to stop at any point in time if a fresh directive isn't issued -- the process only continues as long as each new document causes it to continue. Thus, the content creator has total control.

The interval can be 0 seconds! This will cause the browser to load the new data as soon as it possibly can (after the current data is fully displayed).

The data that is retrieved can be of any type: an image, an audio clip, whatever. One fun thing to envision is 0-second continuous updating of a live image (e.g. a camera feed), or a series of still images. Poor man's animation, kind of. Netscape Communications are considering mounting a camouflaged IndyCam on the prow of Jim Clark's boat and feeding live images to the world using this mechanism.

A "Refresh" header can be returned as part of any HTTP response, including a redirection. So a single HTTP response can say "go get this URL now, and then go get this other URL in 10 seconds".

This means a continuous random URL generator can be made. Have a normal random URL generator (such as [URouLette](#)) that returns as part of its redirection response a "Refresh" directive that causes the browser to get another random URL from the random URL generator in 18 seconds.

See the impressive URouLette at :
<http://kuhttp.cc.ukans.edu/cwis/organizations/kucia/roulette/roulette.html>

Server Push
<META> Element
JavaScript

Dynamic Documents - Netscape JavaScript

[See Also](#)

NOTE : JavaScript is currently only supported by the **Netscape Navigator** (version 2 and above). Currently, it is under development and is liable to change. For more information on JavaScript, point your browser to <http://home.netscape.com/>. The information provided here only details how to include JavaScript scripts within HTML documents, not how to author actual scripts. Such information is well beyond the scope of this document.

JavaScript is a small, property-based scripting language. Scripts written in JavaScript can enhance the features of client and server applications. For example, a JavaScript script embedded in HTML can recognise and respond to user-initiated events such as form input and page navigation.

E.g.: A page author can write a JavaScript script to verify that numeric information has been entered into a form requesting a telephone number or zip code. Without any network transmission, an HTML script with embedded JavaScript can interpret the entered text and alert the user with an appropriate message dialog.

A script is embedded in HTML within a `<SCRIPT>` element.

```
<SCRIPT>...</SCRIPT>
```

The text of a script is inserted between `<SCRIPT>` and its end element.

Attributes within the `<SCRIPT>` element can be specified as follows:

```
<SCRIPT LANGUAGE="JavaScript">  
</SCRIPT>
```

The **LANGUAGE** attribute is mandatory unless the **SRC** attribute is present and specifies the scripting language.

```
<SCRIPT SRC="http:common.JavaScript">  
</SCRIPT>
```

The **SRC** attribute is optional and, if given, specifies a URL that loads the text of a script.

```
<SCRIPT LANGUAGE="language" SRC=url>
```

Both attributes may be present.

Usage Notes

Scripts placed within `<SCRIPT>` elements are evaluated immediately upon loading of the page. Functions are stored, but not executed. Functions are executed by events in the page. **SRC** URL information is read in and evaluated immediately as script container content. **SRC** script is evaluated before in-page script. A named `<SCRIPT>` element contains a function body that can be called in an **ONCHANGE** event or other event-handler attribute. Scripts may be placed inside comment fields to ensure that the script is not displayed when the page's HTML is viewed with a browser unaware of the `<SCRIPT>` element. The entire script is encased by HTML comment elements:

```
<!-- Begin to hide script contents from old browsers.  
<!-- End the hiding here.-->
```

Except for string literals, JavaScript is case-insensitive and treats all letters as lowercase. Single

quotes may be used in JavaScript so that scripts can be distinguished from attribute values using double quotes.

For more information about scripts, the language and some examples, see [**http://home.netscape.com**](http://home.netscape.com)

Server Push
<META> Element
Client Pull

<EMBED> - Embedded Objects

NOTE : This tag is **only** recognised by the **Windows** version of Netscape Navigator, versions 1.1 and above.

The <EMBED> element allows you to put documents directly into an HTML page, somewhat like OLE capabilities of Microsofts [Internet Assistant](#)

The syntax is:

```
<EMBED SRC="images/embed.bmp">
```

The <EMBED> element will allow you to embed documents of any type. Your user only needs to have an application which can view the data installed correctly on their machine.

If a width and height are specified, the embedded object is scaled to fit the available space. For example this is the same bitmap as above, scaled:

```
<EMBED SRC="images/embed.bmp" WIDTH=250 HEIGHT=50>
```

Embedded objects can be activated by double clicking them in the Netscape window. The application that supports use of the embedded object will be launched, with the object present.

e.g. If a .BMP graphic file was embedded into a document, when the user double clicks the picture, Netscape will launch PaintBrush (unless .BMP is associated with another application) with the object loaded, ready for editing.

NOTE : Using the <EMBED> element, you should be sure that the user will have a suitable application available that is OLE compliant (unless of course, the embedded file is a data type that will be handled by a plug-in module - see below). Otherwise, the HTML document will not be displayed as hoped. Essentially this element produces the same results as embedding objects in Word for Windows - the object is displayed, and can be edited in a suitable application by double clicking on the object. The display of the embedded object in Netscape is then changed (including any extra information added in the edit), but the actual file remains unchanged.

Netscape Plug-ins

Netscapes plug-ins will make use of the <EMBED> element. Essentially, plug-ins are dynamic code modules which are associated with a MIME data type that the Netscape client has no native support for. When Netscape encounters an unknown data type from a server, it will search for a plug-in that is associated with that MIME type and load it, enabling viewing/transforming of the data object.

Plug-ins can have one of three modes of operation. They can be embedded, full-screen or hidden.

An embedded plug-in is a part of a larger HTML document, where the plug-in is to be visible as a rectangular sub-part of an HTML page. This is similar to an embedded GIF or JPG image, or an AVI video (see [In-Line video](#)) today. Indeed, an example of such a plug-in could be an AVI video player, where the video is played in a viewing rectangle in the HTML document and can be made to respond to user actions. (An example of an AVI playing plug-in is currently available as beta code from the Netscape web site - <http://home.netscape.com>)

A full-screen plug-in would be the viewer for a particular data type that was not part of an HTML document. In this mode, a plug-in will completely fill the inner frame of a Netscape window with its representation of some data type. An example of this type of plug-in would be an Adobe Acrobat viewer to view .PDF (portable document format)

A hidden plug-in is one that needs no user visible elements. An example of this might be a MIDI player, or a decompression engine.

At the time of writing, only a few plug-in modules had been released for Netscape. Paper Software Inc. released WebFX, a VRML plug-in for Netscape 2.0. This plug-in uses the `<EMBED>` element, together with the **BORDER**, **ALIGN**, **WIDTH**, **HEIGHT** and **SRC** attributes, allowing the embedding of VRML 'world' objects into HTML documents. The `ALIGN` attribute supports values of left or right, aligning the embedded VRML object on the page. `BORDER` sets the border width to be displayed, defaulting with no border. `WIDTH` and `HEIGHT` attributes are similar to those used in the element, except that when used separately, the VRML object will be uniformly scaled, while the embedding window is scaled according to the value of either attribute. Hence it is possible to create a 'post box' type display of a VRML world. Simply using `<EMBED SRC="filename.wrl">` results in the VRML object being displayed in a small rectangle 50x50 pixels. To employ WebFX as a full-screen plug-in, it is necessary to include the VRML object in the URL path of an <A HREF...> hypertext link

For more information about plug-ins, see the Netscape web site. <http://home.netscape.com/>

Microsoft Internet Assistant for Word for Windows turns Word into a fully functional World Wide Web browser/HTML editor. HTML document editing is possible within Word, which - at the click of the mouse - becomes a Web browser. It supports use of all HTML 2.0 elements and some of the HTML 3.0 elements currently supported by other browsers (i.e. Netscape/Mosaic) and the Internet Explorer. For more information, point your browser to <http://www.microsoft.com/> For the Windows'95 specific version (for Word 7), see <http://www.windows.microsoft.com/>

Quick Reference

A
B
C
D
E
F
G
H
I
J
K
L
M
N
O
P
Q
R
S
T
U
V
W
X
Y
Z

Exit

<!-- ...
<!--# ...
<!DOCTYPE ...>

<A ...>
<ADDRESS>
<APPLET ...>

<BASE ...>
<BASEFONT SIZE= ...>
<BG SOUND ...>
<BIG>
<BLINK>
<BLOCKQUOTE>
<BODY>

<CAPTION>

<CENTER>

<CITE>

<CODE>

<DD>

<DIR>

<DIV>

<DL>

<DT>

<EMBED ...>

<FORM>

<FRAME ...>

<FRAMESET ...>

<H ALIGN= ...>

<H1>

<H2>

<H3>

<H4>

<H5>

<H6>

<HEAD>

<HR>

<HTML>

<I>

<INPUT>

<ISINDEX>

<KBD>

<LINK ...>

<MAP ...>

<MARQUEE ...>

<MENU>

<META ...>

<NEXTID ...>

<NOBR>

<NOFRAMES>

<OPTION>

<P>

<PRE>

<SAMP>
<SCRIPT ...>
<SELECT>
<SMALL>
<SOUND ...>
<STRIKE>

<SUB>
<SUP>

<TABLE ...>
<TD>
<TEXTAREA>
<TH>
<TITLE>
<TR>
<TT>

<U>

<VAR>

<WBR>

<!--# ...
...echo
...include
...fsize
...flastmod
...exec
...config
...odbc
...email
...if
...goto
...label
...break

<BODY ...>
...BACKGROUND
...TEXT
...LINK
...VLINK
...ALINK
...BGCOLOR
...BGPROPERTIES
...LEFTMARGIN
...TOPMARGIN

<A ...>
...HREF
Mailto : ...TITLE
...NAME
...TITLE
...REL
...REV
...URN
...METHODS
...TARGET

<APPLET ...>

...CODEBASE

...CODE

...ALT

...NAME

...WIDTH/HEIGHT

...ALIGN

...VSPACE/HSPACE

...PARAM NAME/VALUE

<FRAME...>

...SRC

...NAME

...MARGINWIDTH

...MARGINHEIGHT

...SCROLLING

...NORESIZE

<FRAMESET...>
...ROWS
...COLS

<HR...>

...SIZE

...WIDTH

...ALIGN

...NOSHADE

<DL...>

<DT>

<DD>

...COMPACT

<OL...>

...TYPE

...START

...VALUE

...ALIGN

...ALT

...ISMAP

...SRC

...WIDTH

...HEIGHT

...BORDER

...VSPACE

...HSPACE

...LOWSRC

...USEMAP

...VRML

<FONT...>

...SIZE

...COLOR

...FACE

<MAP...>

...SHAPE

...COORDS

...AREA

<META ...>
...HTTP-EQUIV
...NAME
...CONTENT

<INPUT...>

...ALIGN

...CHECKED

...MAXLENGTH

...NAME

...SIZE

...SRC

...TYPE :

CHECKBOX; HIDDEN; IMAGE;

PASSWORD; RADIO; RESET;

SUBMIT; TEXT; TEXTAREA

...VALUE

<TEXTAREA...>

...NAME

...ROWS

...COLS

...WRAP

<TABLE...>

...BORDER

...CELLSPACING

...CELLPADDING

...WIDTH

...ALIGN

...VALIGN

...BGCOLOR

...BORDERCOLOR

...BORDERCOLORLIGHT

...BORDERCOLORDARK

<TR...>

...ALIGN

...VALIGN

...BGCOLOR

...BORDERCOLOR

...BORDERCOLORLIGHT

...BORDERCOLORDARK

<TD...>

...ROWSPAN

...COLSPAN

...ALIGN

...VALIGN

...NOWRAP

...BGCOLOR

...BORDERCOLOR

...BORDERCOLORLIGHT

...BORDERCOLORDARK

<TH...>

...ROWSPAN

...COLSPAN

...ALIGN

...VALIGN

...NOWRAP

...BGCOLOR

...BORDERCOLOR

...BORDERCOLORLIGHT

...BORDERCOLORDARK

<SCRIPT...>
...LANGUAGE
...SRC

<MARQUEE...>

...ALIGN

...BEHAVIOR

...BGCOLOR

...DIRECTION

...HEIGHT

...WIDTH

...HSPACE

...LOOP

...SCROLLAMOUNT

...SCROLLDELAY

...VSPACE

Contacting the Author

After gathering together information about the specification of HTML (level 2.0 and a sketchy level 3.0 available at the time of writing) I realised that no concise (useable) reference existed (apart from the (Internet Engineering Task Force) IETF HTML-working group Internet Drafts and RFC documents), so I decided to construct a HTML reference guide.

This is said reference.

While I take credit (and criticism) for the actual construction/layout of this reference, I cannot take any responsibility for it's content. The original HTML document type was designed by **Tim Berners-Lee** at CERN in 1990. In 1992, **Dan Connolly** wrote the HTML Document Type Definition (DTD) and a brief HTML specification. Since 1993, a wide variety of people have contributed to the evolution of the HTML specification and in 1994, Dan Connolly and Karen Olson Muldrow rewrote the HTML Specification. A complete list of those contributors can be found in the HTML specification, which is now written by **Dave Raggett** and can be obtained from :

<http://www-uk.hpl.hp.co.uk/people/dsr/>

It should be noted that, while this index of HTML elements was complete at the time of writing (covering all HTML elements supported by commonly available browsers), the World Wide Web initiative is evolving at an exponential rate, hence this reference can only be considered a *work in progress*, parallel to the actual specifications themselves. For more information, the archives of the IETF HTML working group discussions should be consulted. Among other sites, the HTML 3 draft specification can be found at :

<ftp://src.doc.ic.ac.uk/computing/internet/internet-drafts/draft-ietf-html-spec-XX.txt>

Therefore, while I am willing to accept [comments and criticisms](#) of the document, I cannot take responsibility for the status of the specification contained within. I will however, endeavour to keep up with the specifications and update this document when necessary.

The information contained herein about Server Push and Client Pull (Dynamic Documents) and the example table HTML is © 1995 Netscape Communications.

Portions reprinted with permission granted by Microsoft Corporation.

The Server Side Includes section of this document is taken from information kindly provided by Mark West of Questar Microsystems Inc.

NOTE : The most recent version of this reference can always be found at <ftp.swan.ac.uk> in the directory [pub/in.coming/htmlib](ftp://pub/in.coming/htmlib). A mailing list is also kept of those people who have expressed an interest in the reference and will be used to notify them of future releases. To subscribe to the mailing list, send mail to the [author](#).

HotJava - Adding an Applet

Attributes

Recent version of the Netscape Navigator (since version 2.0) have added the capability to support *executable content* previously only possible using the actual HotJava browser developed by Sun Microsystems. This allows HTML authors to include live audio, animation, or applications in the form of HotJava applets. Such applets are pre-compiled in the HotJava programming language and included in HTML documents.

NOTE : The information provided here only describes the necessary HTML elements that allow pre-compiled HotJava applets to be added to your HTML documents. It does not describe how to actually write HotJava applets. Such information is well beyond the scope of this document. More information can be found about writing HotJava applets and how to obtain a copy of the HotJava browser from :

<http://java.sun.com/>

Also, the information provided here is relevant for the pre-beta development version of HotJava. Existing applets (developed under the 1.0alpha 3 version) can be *upgraded* to this new version. More details can be found at the Sun Microsystems Web site (URL above).

To add an applet to an HTML page, you need to use the <APPLET> HTML element.

```
<APPLET CODE="MyApplet.class" WIDTH=100 HEIGHT=140>
</APPLET>
```

This tells the viewer or browser to load the applet whose compiled code is in MyApplet.class (in the same directory as the current HTML document), and to set the initial size of the applet to 100 pixels wide and 140 pixels high.

Below is a more complex example of an APPLET element:

```
<APPLET CODEBASE="http://java.sun.com/JDK-prebeta1/applets/NervousText" CODE="NervousText.class"
width=400 height=75 align=center >
<PARAM NAME="text" VALUE="This is the Applet Viewer.">
<BLOCKQUOTE>
<HR>
If you were using a Java(tm)-enabled browser, you would see dancing text instead of this paragraph.
<HR>
</BLOCKQUOTE>
</APPLET>
```

This tells the viewer or browser to load the applet whose compiled code is at the URL <http://java.sun.com/JDK-prebeta1/applets/NervousText/NervousText.class>, to set the initial size of the applet to 400x75 pixels, and to align the applet in the centre of the line. The viewer/browser must also set the applet's "text" attribute (which customizes the text this applet displays) to be "This is the Applet Viewer." If the page is viewed by a browser that can't execute applets written in the Java Programming Language, then the browser will ignore the APPLET and PARAM elements, displaying the HTML between the <BLOCKQUOTE> and </BLOCKQUOTE> elements.

Java(tm)-enabled browsers *ignore* that HTML.

Here's the complete syntax for the APPLET element:

```
<APPLET
[CODEBASE = codebaseURL]
```

```
CODE = appletFile
[ALT = alternateText]
[NAME = appletInstanceName]
WIDTH = pixels HEIGHT = pixels
[ALIGN = alignment]
[VSPACE = pixels] [HSPACE = pixels]
>
[<PARAM NAME = appletAttribute1 VALUE = value>]
[<PARAM NAME = appletAttribute2 VALUE = value>]
...
[alternateHTML]
</APPLET>
```

CODEBASE = *codebaseURL*

This optional attribute specifies the base URL of the applet -- the directory that contains the applet's code. If this attribute is not specified, then the document's URL is used.

CODE = *appletFile*

This required attribute gives the name of the file that contains the applet's compiled Applet subclass. This file is relative to the base URL of the applet. It cannot be absolute.

ALT = *alternateText*

This optional attribute specifies any text that should be displayed if the browser understands the `APPLET` element but can't run applets written in the Java(tm) Programming Language.

NAME = *appletInstanceName*

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

WIDTH = *pixels* **HEIGHT** = *pixels*

These required attributes give the initial width and height (in pixels) of the applet display area, not counting any windows or dialogs that the applet brings up.

ALIGN = *alignment*

This required attribute specifies the alignment of the applet. The possible values of this attribute are the same as those for the `IMG` element: left, right, top, texttop, middle, absmiddle, baseline, bottom, absbottom.

VSPACE = *pixels* **HSPACE** = *pixels*

These option attributes specify the number of pixels above and below the applet (`VSPACE`) and on each side of the applet (`HSPACE`). They're treated the same way as the `IMG` element's `VSPACE` and `HSPACE` attributes.

<PARAM NAME = *appletAttribute1* VALUE = *value*>

This element is the only way to specify an applet-specific attribute. Applets access their attributes with the `getParameter()` method.

The following attributes are used in the <APPLET> element for the inclusion of HotJava applets in HTML documents.

CODEBASE

CODE

ALT

NAME

WIDTH/HEIGHT

ALIGN

VSPACE/HSPACE

PARAM NAME/VALUE

Frames - Advanced page formatting

[Attributes](#)

[See Also](#)

NOTE : The use of Frames is currently **only** supported by recent versions of the Netscape Navigator (from version 2.0)

Frames extend the layout flexibility of web pages by allowing the visible client area to be divided into more than one sub-region. Each sub-region, or frame, has several properties:

- 1) It can load its own URL, independently of the other frames.
- 2) It can be given a NAME, allowing it to be targeted by other URL's
- 3) It resizes itself dynamically in response to changes in the size of the visible client area, and it can choose to allow or disallow itself to be manually resized by the user.

These properties enable a number of new scenarios:

1. Information that the page designer wishes to remain constant and visible, such as Toolbar-like UI, Confidentiality messages, Title graphics, can be put on a frame separate from the rest of the site. As the user moves through the site in the "live" frame, the "static" frame is always visible, with no redraw.
2. Table of Contents scenarios become much clearer. A "left" frame could contain a set of links, each of which when clicked targets its results into the "right" frame.
3. A "query" frame could contain form HTML for the submission of database queries, while a "results" frame could receive the results of each query.

Frames are generated by three things: FRAMESET tags, FRAME tags, and FRAME Documents.

Frame Document

A Frame document has a basic structure very much like your normal HTML document, except the BODY container is replaced by a FRAMESET container which describes the sub-HTML documents, or Frames, that will make up the page.

```
<HTML>
<HEAD>
</HEAD>
<FRAMESET>

</FRAMESET>
</HTML>
```

Frame Syntax

Frame syntax is similar in scope and complexity to that used by tables, and has been designed to be quickly processed by Internet client layout engines.

<FRAMESET>

This is the main container for a Frame. It has 2 attributes ROWS and COLS. A frame document has no BODY, and no tags that would normally be placed in the BODY can appear before the FRAMESET tag, or the FRAMESET will be ignored. The FRAMESET tag has a matching end tag, and within the FRAMESET you can only have other nested FRAMESET tags, FRAME tags, or the NOFRAMES tag.

ROWS="row_height_value_list"

The ROWS attribute takes as its value a comma separated list of values. These values can be absolute pixel values, percentage values between 1 and 100, or relative scaling values. The number of rows is implicit in the number of elements in the list. Since the total height of all the rows must equal the height of

the window, row heights might be normalised to achieve this. A missing `ROWS` attribute is interpreted as a single row arbitrarily sized to fit.

Syntax of value list.

`value`

A simple numeric value is assumed to be a fixed size in pixels. This is the most dangerous type of value to use since the size of the viewer's window can and does vary substantially. If fixed pixel values are used, it will almost certainly be necessary to mix them with one or more of the relative size values described below. Otherwise the client engine will likely override your specified pixel value to ensure that the total proportions of the frame are 100% of the width and height of the user's window.

`value%`

This is a simple percentage value between 1 and 100. If the total is greater than 100 all percentages are scaled down. If the total is less than 100, and relative-sized frames exist, extra space will be given to them. If there are no relative-sized frames, all percentages will be scaled up to match a total of 100%.

`value*`

The value on this field is optional. A single "*" character is a "relative-sized" frame and is interpreted as a request to give the frame all remaining space. If there exist multiple relative-sized frames, the remaining space is divided evenly among them. If there is a value in front of the "*", that frame gets that much more relative space. "2*," would give 2/3 of the space to the first frame, and 1/3 to the second.

Example for 3 rows, the first and the last being smaller than the centre row:

```
<FRAMESET ROWS="20%, 60%, 20%">
```

Example for 3 rows, the first and the last being fixed height, with the remaining space assigned to the middle row:

```
<FRAMESET ROWS="100, *, 100">
```

COLS="column_width_list"

The `COLS` attribute takes as its value a comma separated list of values that is of the exact same syntax as the list described above for the `ROWS` attribute.

The `FRAMESET` tag can be nested inside other `FRAMESET` tags. In this case the complete subframe is placed in the space that would be used for the corresponding frame if this had been a `FRAME` tag instead of a nested `FRAMESET`.

<FRAME>

This tag defines a single frame in a frameset. It has 6 possible attributes: `SRC`, `NAME`, `MARGINWIDTH`, `MARGINHEIGHT`, `SCROLLING`, and `NORESIZE`. The `FRAME` tag is not a container so it has no matching end tag.

SRC="url"

The `SRC` attribute takes as its value the URL of the document to be displayed in this particular frame. `FRAMES` without `SRC` attributes are displayed as a blank space the size the frame would have been.

NAME="window_name"

The `NAME` attribute is used to assign a name to a frame so it can be targeted by links in other documents (These are usually from other frames in the same document.) The `NAME` attribute is optional; by default all windows are unnamed.

Names must begin with an alphanumeric character. However, several reserved names have been defined, which start with an underscore.

These are currently:

| | |
|---------------------|---|
| <code>_blank</code> | Always load this link into a new, unnamed window. |
| <code>_self</code> | Always load this link over yourself. |

`_parent` Always load this link over your parent. (becomes self if you have no parent).

`_top` Always load this link at the top level. (becomes self if you are at the top).

NOTE : Although these are reserved names for the `NAME` attribute of the `<FRAME>` element, they should only be referred to using an [Anchor Target](#). That is, used to target specific windows, allowing smoother transition between framed documents and between framed and *normal* documents.

MARGINWIDTH="value"

The `MARGINWIDTH` attribute is used when the document author wants some control of the margins for this frame. If specified, the value for `MARGINWIDTH` is in pixels. Margins can not be less than one-so that frame objects will not touch frame edges-and can not be specified so that there is no space for the document contents. The `MARGINWIDTH` attribute is optional; by default, all frames default to letting the browser decide on an appropriate margin width.

MARGINHEIGHT="value"

The `MARGINHEIGHT` attribute is just like `MARGINWIDTH` above, except it controls the upper and lower margins instead of the left and right margins.

SCROLLING="yes|no|auto"

The `SCROLLING` attribute is used to describe if the frame should have a scrollbar or not. Yes results in scrollbars always being visible on that frame. No results in scrollbars never being visible. Auto instructs the browser to decide whether scrollbars are needed, and place them where necessary. The `SCROLLING` attribute is optional; the default value is auto.

NORESIZE

The `NORESIZE` attribute has no value. It is a flag that indicates that the frame is not resizable by the user. Users typically resize frames by dragging a frame edge to a new position. Note that if any frame adjacent to an edge is not resizable, that entire edge will be restricted from moving. This will effect the resizability of other frames. The `NORESIZE` attribute is optional; by default all frames are resizable.

<NOFRAMES>

This tag is for content providers who want to create alternative content that is viewable by non-Frame-capable clients. A Frame-capable Internet client ignores all tags and data between start and end `NOFRAMES` tags.

Frames are controlled by the following elements and their attributes:

<FRAMESET>

ROWS

COLS

<FRAME>

SRC

NAME

MARGINWIDTH

MARGINHEIGHT

SCROLLING

NORESIZE

<NOFRAMES>

A Frames example

A Frames example

For the purposes of this example, a portion of this HLP reference was formatted into HTML, using frames.

The example actually consists of six different HTML documents. These are detailed below:

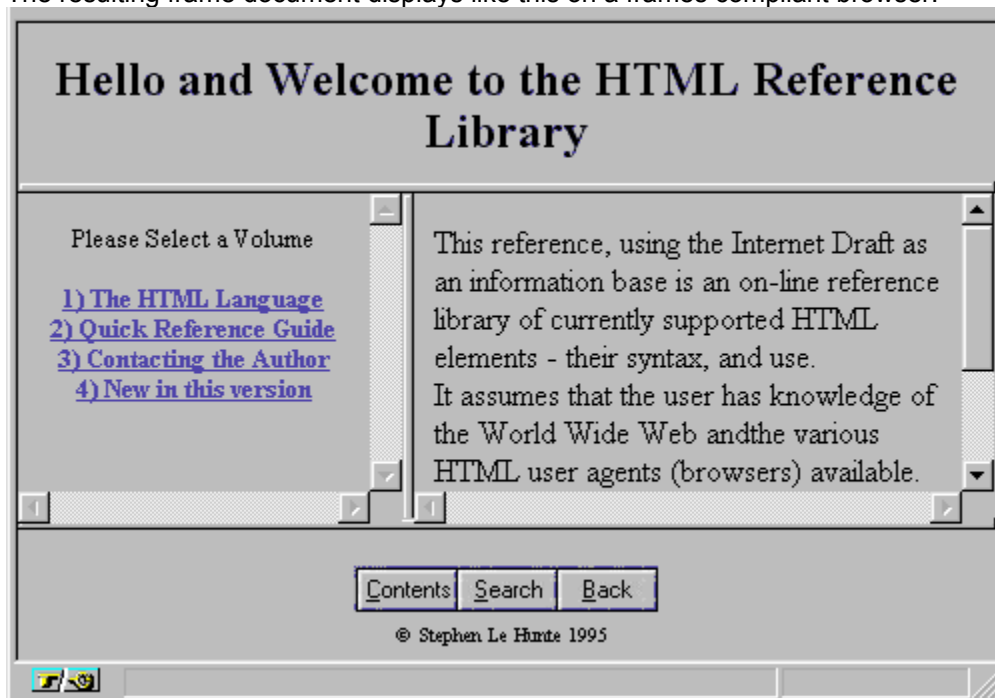
- 1) [The main frame set up document](#)
- 2) [The title document](#)
- 3) [The contents document](#)
- 4) [The main text document](#)
- 5) [The navigation buttons document](#)
- 6) [The HTML language document](#)

Documents 2, 4, 5 and 6 have no Markup relevant to the use of Frames and have only been included here for reasons of completeness.

To see how the document displays on a frames compliant browser (i.e. Netscape 2.0 and above), click the camera:



The resulting frame document displays like this on a frames compliant browser:



The main frame set up document

The main document that sets up the example frame is as follows:

```
<HTML>
<!--HTMLLIB.HTM-- >
<HEAD>
<TITLE>The HTML Reference Library</TITLE>
</HEAD>
<BASEFONT SIZE=3>

<FRAMESET ROWS="85,*,65">
<FRAME SCROLLING="no" NAME="title" NORESIZE SRC="title.htm">
<FRAMESET COLS="40%,60%">
<FRAME SCROLLING="yes" NAME="toc" SRC="toc.htm">
<FRAME SCROLLING="yes" NAME="main page" SRC="main.htm">
</FRAMESET>
<FRAME SCROLLING="no" NAME="HLP buttons" NORESIZE SRC="buttons.htm">

<NOFRAME>

</NOFRAME>
</FRAMESET>
```

A line by line breakdown

```
<FRAMESET ROWS="85,*,65">
```

This line divides the page into three regions, the top region being 85 pixels in height, the bottom region being 65 pixels in height, the middle region occupying the rest of the Netscape window.

```
<FRAME SCROLLING="no" NAME="title" NORESIZE SRC="title.htm">
```

This line sets the top region of the window (the region that is 85 pixels high) to be a non-scrolling, non-resizable region. Its name is title (so, any other link that specifies "title" with its TARGET attribute would be displayed in this region.)

```
<FRAMESET COLS="40%,60%">
```

This splits the middle region of the Netscape window into two sections horizontally. The left hand section being 40% of the window width, the right hand section being the remaining 60% of the window width.

```
<FRAME SCROLLING="yes" NAME="toc" SRC="toc.htm">
```

```
<FRAME SCROLLING="yes" NAME="main page" SRC="main.htm">
```

These two lines (as the other FRAME line above) set the attributes for the two middle sections of the page. That is, it names the regions "toc" and "main page" respectively and links to the two pages to be displayed in the regions.

```
</FRAMESET>
```

This line closes the sub-frames that were opened in the middle section of the main framed regions.

```
<FRAME SCROLLING="no" NAME="HLP buttons" NORESIZE SRC="buttons.htm">
```

This line defines the properties of the remaining main region of the window - namely the bottom region, 65 pixels high. It defines it as a non-scrolling, non-resizable region (ideal for navigation tools)

The use of the NAME attribute in setting up the page regions is so that each different page can have linked pages sent to it by name. This way, where the document called (when a link is activated) is displayed can be controlled by the author.

NOTE : Here, no information has been provided between the `<NOFRAME>` ... `</NOFRAME>` elements. This is where the author must put information that s/he wishes to be displayed on browsers that do not support the use of frames.



The title document

NOTE : This document contains no Markup relevant to the use of the frames, but has been included for reasons of completeness.

This document is the Title for the paged document. It resides in the top frame, which is a non-scrolling non-resizeable frame. Hence the title will always be displayed in the same place. Note that for frame sub-documents titles are not required. The title of the site will always be taken from the main frame page.

```
<HTML>
<!--TITLE.HTM-- >
<BODY>
<BASEFONT SIZE=3>
<CENTER>
<H2 ALIGN=center>Hello and Welcome to the HTML Reference Library</H2>
<BR>
</CENTER>
</BODY>
</HTML>
```



The contents document

This is the Table of Contents page. It appears on the left scrolling frame region. This section has been used (in this example) for a stationary table of contents.

```
<HTML>
<!--TOC.HTM-- >
<BODY>
<BASEFONT SIZE=2>
<CENTER>
Please Select a Volume<BR><BR>
  <A HREF="lang.htm" TARGET="main page"><B>1) The HTML Language</B></A><BR>
  <A HREF="qr.htm" TARGET="main page"><B>2) Quick Reference Guide</B></A><BR>
  <A HREF="author.htm" TARGET="main page"><B>3) Contacting the Author</B></A><BR>
  <A HREF="new.htm" TARGET="main page"><B>4) New in this version</B></A><BR>
</CENTER>
</BODY>
</HTML>
```

The use of the `TARGET` attribute in the anchor means that when each link is activated the document accessed will be displayed in the frame region named "main page". Thus, any documents accessed from the table of contents will appear in the framed region to the right of the table of contents.



The main text document

NOTE : This document contains no Markup relevant to the use of the frames, but has been included for reasons of completeness.

This document is the document that appears in the right hand framed region of the page the first time the page is accessed.

```
<HTML>
<!--MAIN.HTM-- >
<BODY>
  This reference, using the Internet Draft as an information base is an on-line
  reference library of currently supported HTML elements - their syntax, and
  use.<BR>
  It assumes that the user has knowledge of the World Wide Web and the various HTML
  user agents (browsers) available. Information on specific browsers, or the
  broader topic of 'The World Wide Web' can be obtained by reading the World Wide
  Web FAQ.<BR>
</BODY>
</HTML>
```



The navigation buttons document

NOTE : This document contains no Markup relevant to the use of the frames, but has been included for reasons of completeness.

This document resides at the bottom of the framed document. This region is a non-scrollable non-resizable region. As such, is ideal for a set of navigation buttons or other tools, as these could be. For the purposes of this example, the buttons are just a graphic image.

```
<HTML>
<!--BUTTONS.HTM-- >
<BODY>
<CENTER>
  <IMG SRC="buttons.jpg"><BR>
  <FONT SIZE=1>&copy; Stephen Le Hunte 1995</FONT>
</CENTER>
</BODY>
</HTML>
```



The HTML language document

NOTE : This document contains no Markup relevant to the use of the frames, but has been included for reasons of completeness.

This document is accessed from choosing the first option from the table of contents. When accessed, it would be displayed in the right hand section of the middle regions.

```
<HTML>
<!--LANG.HTM-- >
<BODY>
<CENTER><B>The HTML Language</B></CENTER>
<BR>
The vast range of HTML Markup currently supported by available HTML user agents
(Web browsers, such as Netscape, Mosaic etc.) can be divided into the following
sections. Some elements featured here, may not be supported by all browsers.
Where an element is known to be supported by specific browsers, the element
description will be labelled as such.<BR>
</BODY>
</HTML>
```



Document Sound

NOTE : Two different elements now exist for employing in-line sound in a HTML document. The first is **BGSOUND**, this element is currently only supported by the Microsoft **Internet Explorer**. The other is **SOUND**, which is currently only supported by **NCSA Mosaic**. Mosaic does also support a limited version of Microsoft's **BGSOUND** element. ([see below](#))

The new **BGSOUND** tag allows you to create pages with background sounds or "soundtracks." Sounds can either be samples (.WAV or .AU format) or MIDI (.MID format).

The HTML used to insert a background sound into a page is:

```
<BGSOUND SRC="start.wav">
```

The attributes associated with the **BGSOUND** element are **SRC** and **LOOP**.

SRC

This attribute specifies the address of a sound to be played.

LOOP

This attribute specifies how many times a sound will loop when activated. If **n=-1** or **LOOP=INFINITE** is specified, the sound will loop indefinitely.

Examples

```
<BGSOUND SRC="boing.wav">
```

This would play the specified WAV file as soon as Internet Explorer has downloaded the file.

```
<BGSOUND SRC="boing.wav" LOOP=INFINITE>
```

This would play the specified WAV file as soon as it has finished being loaded and would continuously play the file until another page is loaded.

NCSA Mosaic supports use of the **SOUND** element for playing in-line sound. This element allows the playing of *.WAV files in pages. **NOTE** : The **SOUND** element is only supported by **Mosaic**

The syntax is:

```
<SOUND SRC="filename.wav">
```

This element is not currently supported by the HTML working group, although the Mosaic authors believe it will be included. Inline sound files can be placed in any part of the document and Mosaic will act on the sound file when its position is visible to the document view window. Sound files can be implemented as background sounds using the **LOOP** attribute. An HTML author can also delay the play of an inline sound for x number of seconds using the **DELAY** attribute.

The Attributes of the sound tag are:

```
LOOP=infinite and DELAY=sec.
```

Examples:

```
<SOUND SRC="*.wav" LOOP=infinite>
```

```
<SOUND SRC="*.wav" DELAY=10>
```

NOTE : Although Mosaic will support the use of the BGSOUND element, it will not play in-line *.MID MIDI files. For this, it will launch an external application. It will play *.WAV files using the BGSOUND element though.

Highlighted Scrolling Text

Attributes

NOTE : This element is currently only supported by the Microsoft **Internet Explorer**.

The new `<MARQUEE>` element allows the author to create a scrolling text marquee (as the name suggests, a scrolling text region much like the Windows Marquee screen saver).

Marquees can be left- or right-aligned, like images and have a variety of attributes to control them.

ALIGN

This attribute can be set to either `TOP`, `MIDDLE` or `BOTTOM` and specifies that the text around the marquee should align with the top, middle, or bottom of the marquee.

BEHAVIOR

This can be set to `SCROLL`, `SLIDE` or `ALTERNATE`. It specifies how the text should behave. `SCROLL` (the default) means start completely off one side, scroll all the way across and completely off, then start again. `SLIDE` means start completely off on side, scroll in and stop as soon as the text touches the other margin. `ALTERNATE` means bounce back and forth within the marquee.

BGCOLOR

This specifies a background colour for the marquee, either as a `rrggb` hex triplet, or as one of the Internet Explorer prenamed colours. (See [](#) for more information)

DIRECTION

This specifies in which direction the text should scroll. The default is `LEFT`, which means scrolling to the left from the right. This attribute can also be set to `RIGHT`, which would cause the marquee to scroll from the left to the right.

HEIGHT

This specifies the height of the marquee, either in pixels (`HEIGHT=n`) or as a percentage of the screen height (`HEIGHT=n%`).

WIDTH

This specifies the width of the marquee, either in pixels (`WIDTH=n`) or as a percentage of the screen height (`WIDTH=n%`).

HSPACE

This specifies left and right margins for the outside of the marquee, in pixels.

LOOP

`LOOP` specifies how many times a marquee will loop when activated. If `n=-1`, or `LOOP=INFINITE` is specified, the marquee will loop indefinitely.

SCROLLAMOUNT

Specifies the number of pixels between each successive draw of the marquee text. That is, the amount for the text to move between each draw.

SCROLLDELAY

`SCROLLDELAY` specifies the number of milliseconds between each successive draw of the marquee text. That is, it controls the speed at which text draw takes place.

VSPACE

`VSPACE` specifies the top and bottom margins for the outside of the marquee, in pixels.

Examples

```
<MARQUEE>This text will scroll from left to right slowly</MARQUEE>
```

```
<MARQUEE ALIGN=TOP>The following words, "Hi there!", will be aligned with the top  
of this marquee.</MARQUEE> Hi there!
```

```
<MARQUEE BEHAVIOR=SLIDE>This marquee will scroll in and "stick."</MARQUEE>
```

```
<MARQUEE HEIGHT=50% WIDTH=80%>This marquee, is half the height of the screen and  
80% of the screen width.</MARQUEE>
```

```
<MARQUEE SCROLLDELAY=5 SCROLLAMOUNT=50>This is a very fast marquee.</MARQUEE>
```


The following attributes are supported within the <MARQUEE> element.

ALIGN
BEHAVIOR
BGCOLOR
DIRECTION
HEIGHT
WIDTH
HSPACE
LOOP
SCROLLAMOUNT
SCROLLDELAY
VSPACE

This element is **Internet Explorer** specific.

New in this version

This version of the HTML Reference Library contains mostly updated information and aesthetic/functional changes. There is a lot of new information though.

The [Quick Reference](#) section is now displayed in a separate window. This allows the Quick Reference section to be kept open for as long as required. It is now accessed through the **Quick Ref.** Button on the button bar at the top of the main window, or by pressing Alt+Q. Also, in the Quick Reference section, with elements that accept many attributes that don't fit into the initial topic display, a list of the attributes are displayed in a pop-up box from the Quick Reference entry, allowing quicker targeting of desired attribute information.

Here are the element/attribute changes/additions detailed in this version.

[Reserved names](#) for the <FRAME> element.

[<APPLET>](#) element replaces the previous <APP> element.

[Netscape JavaScript](#) elements.

Netscapes [plug-ins](#). Description and some detail of a VRML plug-in module.

[Server Side Includes](#) (from the SSI+ 1.0 specification).

[LEFTMARGIN](#) and [TOPMARGIN](#) attributes for the <BODY> element - Allow margin setting for the Internet Explorer.

[BGCOLOR](#), [BORDERCOLOR](#), [BORDERCOLORLIGHT](#) and [BORDERCOLORDARK](#) attributes for flexible colouring of tables using the Internet Explorer. There are also many new alignment options for various <TABLE> related elements, supported by the new Internet Explorer.

[Embedding of VRML worlds](#) (and ActiveVRML animated objects) supported by Microsofts Virtual Explorer.

New in previous versions.

Version 2.0

This version of the HTML Reference Library contained probably the most new information of any release. This section was an addition. Here are quick links to the elements or attributes that were new to version 2.0 (released just after Netscape 2.0, Internet Explorer 2.0 and Mosaic 2.0)

New Elements

[In-line video](#) support for Internet Explorer

Document [sound](#) capabilities of Internet Explorer and Mosaic

Highlighted scrolling [Marquee](#) text - Internet Explorer

[HotJava](#) applet inclusion - Netscape

[Frames](#) - Advanced page formatting.

The [<DIV>](#) element. Text alignment

New Attributes

[Watermarking](#) now available in Internet Explorer

[Floating tables](#)

[Colouring of table cells](#)

[Client Pull](#) is now supported by Internet Explorer

[Targetted windows](#) and target setting of [base](#) pages

[Font colouring](#) now supported by Netscape

Netscape now supports [Client Side Image Maps](#)

[Wrapping the text in a TEXTAREA box](#) on a form is now available in Netscape



The HTML Reference Library
Prepared by Stephen Le Hunte, 1995
mailto : cmlehunt@swan.ac.uk

mailto : cmlehunt@swan.ac.uk

SSI+ - Server Side Includes

NOTE : Server Side Includes are HTTP server specific. At present SSI+ has only been implemented in the WebQuest and WebQuest95 Web servers from Questar Microsystems, although the technology has been licensed by Microsoft and Netscape for inclusion in their servers and so should achieve greater prominence in the near future. You should check with the server maintainer to be sure that implementing SSI+ tokens in your HTML documents will have the desired effect.

Server side includes (SSI) applied to an HTML document, provide for interactive real-time features such as echoing current time, conditional execution based on logical comparisons, querying or updating a database, sending an [email](#), etc., with no programming or CGI scripts. An SSI consists of a special sequence of characters (tokens) on an HTML page. As the page is sent from the HTTP server to the requesting client, the page is scanned by the server for these special tokens. When a token is found the server interprets the data in the token and performs an action based on the token data.

The format of a SSI token is as follows :

```
<!--#<tag><variable set> '-->
```

where :

<!--# is the opening identifier, a SSI token always starts with this.

<tag> is one of the following: [echo](#), [include](#), [fsize](#), [flastmod](#), [exec](#), [config](#), [odbc](#), [email](#), [if](#), [goto](#), [label](#), [break](#).

<variable set> is a set of one or more variables and their values. The values allowed here are dependent on the <tag> and are listed below each tag. The format of a variable set is as follows :

```
<variable name> '=' '' variable data '' <variable name2> '=' '' variable data2 '' <variable name n> '=' '' variable data n ''.
```

"-->" is the closing identifier. A SSI token always ends with this.

SSI tokens may contain special tags (subtokens) which are dereferenced before evaluation of the SSI token takes place. Subtokens maybe inserted any place in the SSI token. Subtokens are especially useful when forming [if](#), [odbc](#), [email](#), and [exec](#) tokens that are based on HTML form data returned from a remote client. The format of a subtoken is as follows: '&&'<subtokendata>'&&' where:

'&&' is a reserved character sequence defining the beginning and end of the subtoken.

<subtokendata> is any value allowed in an [echo](#) token.

SSI Documents

An SSI+ document must have a file extension of '.SHT' or '.SHTM'. For the sake of efficiency the HTTP server will only scan those documents with the aforementioned extensions for SSI+ tokens.

SSI Tags, Variables and Data

The following is a list of the currently supported SSI+ tags:

echo tag provides for inserting the data of certain variables into an HTML page.

include tag provides for inserting the contents of a file into the HTML page at the location of the `include` token.

fsize tag provides for inserting the size of a given file into the HTML page at the location of the `fsize` token.

lastmod tag provides for inserting the last modification date of a given file into the HTML page at the location of the `lastmod` token.

exec tag provides for executing an external executable.

config tag provides for setting certain HTML output options.

odbc tag provides for querying and updating ODBC databases.

email tag provides for sending an email whenever an HTML page is accessed or an HTML form is submitted.

if tag provides for conditional execution of SSI operations and conditional printing of HTML text based on logical comparisons.

goto tag provides for jumping to a label token without executing any SSI code or printing any HTML text between the goto token and the label token.

label tag provides a place for a goto or if goto token to jump to.

break tag provides for termination of HTML documents at any point.

'echo' tag

Attributes

The **echo** tag provides for inserting the data of certain variables into an HTML page.

The only variable under the **echo** tag is '**var**'. The data in an echo token is translated into a string that depends on the value in the variable and that string is inserted into the HTML page at the location of the echo token in the HTML page.

Example. The following string in an HTML page :

The Greenwich Mean Time is `<!--#echo var="DATE_GMT"-->` and I am so happy to be here.

would resolve at runtime into something like:

The Greenwich Mean Time is Fri Jul 21 21:24:48 1995 and I am so happy to be here.

The values allowed as variable data are available from two sources: **Form fields** and **environment variables**.

Form Fields are those datum which are available when a POST operation is performed on a SSI enabled HTML document from an HTML form. Each form field may be referenced by name.

Example. Suppose one has an HTML form with several fields defined, one of which is named 'First Name'. When an HTTP client POSTs the form to an SSI enabled HTML document with an echo token: `<!--#echo var="First Name" -->`, the HTML document will then have the contents of the supplied 'First Name' field inserted upon transmission to the client.

Environment variables are a set of datum defined by the local server and remote client and are defined below.

DOCUMENT_NAME This variable is the complete local directory path of the current document.

DOCUMENT_URI This variable is the local path of the current document referenced to the base directory of the webspace.

QUERY_STRING_UNESCAPED This variable is the unescaped query string sent by the client browser, all shell-special characters escaped with \.

DATE_LOCAL This variable is current local date and time.

DATE_GMT This variable is the current Greenwich Mean date and time.

LAST_MODIFIED This variable is the date and time of the last modification of the current document.

REMOTE_ADDR This variable is the IP address of the remote client browser.

QUERY_STRING This variable is the raw query string sent from the remote browser.

SERVER_SOFTWARE This variable is the name of the HTTP server software.

SERVER_NAME This variable is the local computer name of the HTTP server.

GATEWAY_INTERFACE This variable is the name/version of the Common Gateway Interface served on this HTTP server.

SERVER_PROTOCOL This variable is the name/version of HTTP served on this HTTP server.

SERVER_PORT This variable is the IP port the HTTP server is answering on.

REQUEST_METHOD This variable is the method by which the current document was requested.

PATH_INFO This variable is the extra path info that is sent. This information is regarded as virtual (the path is relative to the base directory of the HTTP server).

PATH_TRANSLATED This variable is the 'PATH_INFO' variable translated from virtual to local (physical) disk location.

SCRIPT_NAME This variable is the virtual path of the script being executed.

REMOTE_HOST This variable is the host name of the remote client.

AUTH_TYPE This variable is the authentication method used to validate the remote client.

REMOTE_USER This variable is the user name used to validate authentication from the remote client.

REMOTE_IDENT This variable is the remote user name if supporting RFC931 identification.

CONTENT_TYPE This variable is the content type of the attached information in the case of a POST or PUT.

CONTENT_LENGTH This variable is the length of the attached information in the case of a POST or PUT.

HTTP_ACCEPT This variable is a comma separated list of mime types that are accepted by the remote browser.

HTTP_USER_AGENT This variable is the name of the remote client browser software.

REFERER This variable is the URL of the HTML document which referred the remote client to this document.

FROM This variable is the name (most likely the-mail address) of the remote client user.

FORWARDED This variable is the name of the proxy server through which this document is being processed.

ACCEPT_LANGUAGE This variable lists the human languages that are acceptable to the remote client.

HTTP_COOKIE This variable contains the cookie sent by the remote client.

The 'echo' tag allows two different variable types to be specified. They are:

Form fields

Environment variables

'include' tag

The **include** tag provides for inserting the contents of a file into the HTML page at the location of the include token. The SSI+ include tag is fully recursive, each document inserted may itself contain further SSI+ insert tokens or any other SSI+ tokens.

Virtual

The virtual variable is used to specify a file path/name relative to the base directory of the HTTP server. The format is: 'virtual="<filename>"' where <filename> is a file path/name relative to the base directory of the HTTP server.

File

The file variable is used to specify a file path/name relative to the directory of the current document. The format is: 'file="<filename>"' where <filename> is a file path/name relative to the directory of the current document.

Example. The following token on an HTML document inserts all text and SSI+ tokens from the file <html base directory>'SSI\INSERT.SHT' into the current document before transmission back to the client browser:

```
<!--#include virtual="SSI\INSERT.SHT" -->
```

'fsize' tag

The **fsize** tag provides for inserting the size of the given file into the HTML page at the location of the fsize token.

Virtual

The virtual variable is used to specify a file path/name relative to the base directory of the HTTP server. The format is: 'virtual=""<filename>"" where <filename> is a file path/name relative to the base directory of the HTTP server.

File

The file variable is used to specify a file path/name relative to the directory of the current document. The format is: 'file=""<filename>"" where <filename> is a file path/name relative to the directory of the current document.

Example. The following token on an HTML document inserts the size of file <html base directory>'SSI\INSERT.SHT' into the current document before transmission back to the client browser:

```
<!--#fsize virtual="SSI\INSERT.SHT" -->
```

'flastmod' tag

The **flastmod** tag provides for inserting the size of the given file into the HTML page at the location of the flastmod token.

Virtual

The virtual variable is used to specify a file path/name relative to the base directory of the HTTP server. The format is: 'virtual=""<filename>"" where <filename> is a file path/name relative to the base directory of the HTTP server.

File

The file variable is used to specify a file path/name relative to the directory of the current document. The format is: 'file=""<filename>"" where <filename> is a file path/name relative to the directory of the current document.

Example. The following token on an HTML document inserts the last modification date of file <html base directory>'SSI\INSERT.SHT' into the current document before transmission back to the client browser:

```
<!--#flastmod virtual="SSI\INSERT.SHT" -->
```

'exec' tag

The **exec** tag provides for executing an external executable system command.

Cmd

The `cmd` variable is used to specify the name and command line parameters of any shell executable command. The format is: `'cmd="<exename> <argument list>"'` where `<exename>` is the path and/or name of the executable command and `<argument list>` is the list of command line arguments to send to the executable command. If a full path is not specified then the 'PATH' environment variable of the server will be searched for the executable.

A shell executable is any executable that may be executed on the console. This allows web administrators and authors to use executables that accept command line arguments as an alternative to CGI executables (that accept only environment variables). The output of shell executables may be echoed into the HTML document. See the [config...cmdecho](#) tag for details.

Cgi

The `cgi` variable is used to specify the name a CGI executable (script) relative to the base directory of the HTTP server. The format is: `'cgi="<exename>"'` where `<exename>` is the path and name of the CGI executable relative to the base directory of the web space. The CGI will be executed and any cgi output will be inserted into the current HTML document at the location of the `cgi` token.

'config' tag

Attributes

The **config** tag provides for setting certain HTML output options.

Errmsg

The **errmsg** variable is used to set the error message that gets printed when the SSI+ engine encounters a parsing error or unavailable required data. This variable is retained for compatibility with standard SSI, you may wish to use the **onerr** variable instead.

Timefmt

The **timefmt** variable is used to set the format of [echo..time](#) SSI+ token output.

Sizefmt

The **sizefmt** variable is used to set the format of [echo..size](#) SSI+ token output.

Cmdecho

The **cmdecho** variable is used to set the output option of subsequent [exec..cmd](#) tokens. The format is 'cmdecho'=""<onoroff>" where <onoroff> is either '**ON**' or '**OFF**'.

When the SSI+ parsing engine encounters an [exec..cmd](#) token it executes the command. If the command returns output then that output may be echoed into the HTML document or it may be ignored. The format of the data echoed is dependent on the presence or absence of **config..cmdprefix** and **config..cmdpostfix** tokens in the document. In the absence of **config..cmdprefix** and **config..cmdpostfix** tokens the output will be echoed exactly as returned with no formatting and no special character interpretation. In the presence of **config..cmdprefix** and/or **config..cmdpostfix** tokens the output will be formatted and interpreted. To activate echoing set **cmdecho** to '**ON**' otherwise set it to '**OFF**'. The default is '**OFF**'.

Cmdprefix'

The **cmdprefix** variable is used to set the string prefixed to each line output from subsequent [exec..cmd](#) tokens. The format is 'cmdprefix=""<string>" where <string> is any character string and/or HTML format tags. When the SSI+ parsing engine encounters an [exec..cmd](#) token it executes the command. If the command returns output then that output may be echoed into the HTML document or it may be ignored. If the output is echoed (see '**cmdecho**' above), then each line output from the executable will be prefixed with the string supplied before being echoed into the HTML document.

Cmdpostfix

The **cmdpostfix** variable is used to set the string appended to the end of each line output from subsequent [exec..cmd](#) tokens. The format is 'cmdpostfix=""<string>" where <string> is any character string and/or HTML format tags. When the SSI+ parsing engine encounters an [exec..cmd](#) token it executes the command. If the command returns output then that output may be echoed into the HTML document or it may be ignored. If the output is echoed (see '**cmdecho**' above), then each line output from the executable will be appended with the string supplied before being echoed into the HTML document.

Onerr

The **onerr** variable is used to set the action to be taken when the SSI+ engine encounters an error. The format is 'onerr=""<action>" where <action> is one of the following tags:

[goto](#) causes a jump to a [label](#) token. The format of the goto tag is:

```
goto <label>
```

where <label> is the name of a [label](#) defined in a subsequent [label](#) tag.

'print' causes text to be printed. The format of the print tag is:

```
print "<text>"
```

where <text> is any HTML text or tag.

'error' causes the current **config.error** message to be printed.

[break](#) causes termination of the HTML document transmission to the client.

'errorbreak' causes the current **config.error** message to be printed, and then causes termination of the HTML document transmission to the client.

'printbreak' causes text to be printed, and then causes termination of the HTML document transmission to the client. The format of the printbreak tag is the same as the format of the print tag.

Example. The following token on an HTML document sets the SSI+ error message to '*** ERROR ***'. From this point on when an error occurs in the SSI+ parsing engine the message '*** ERROR ***' will be inserted into the HTML document at the location of the offending SSI+ statement.

```
<!--#config errmsg="*** ERROR ***" -->
```

Example. The following token on an HTML document sets the SSI+ error action to print a message and terminate the document. From this point on when an error occurs in the SSI+ parsing engine the message will be inserted into the HTML document at the location of the offending SSI+ statement, and the document will be terminated.

```
<!--#config onerr="printbreak "Sorry, we encountered an error while processing your document." -->
```

Example. Suppose you wish to create an HTML document that performs a 'PING' operation on address '204.96.64.171' and then [echo](#) the results back to the client browser, with each line echoed as an element in an unnumbered list.

Insert the following lines into your HTML document:

```
<UL>
<!--#config cmdecho="ON" -->
<!--#config cmdprefix="<LI>" -->
<!--#exec cmd="ping 204.96.64.171 -w 20000" -->
</UL>
```

When the document is accessed by a remote browser the output would look something like this:

```
Pinging 204.96.64.171 with 32 bytes of data:
Reply from 204.96.64.171: bytes=32 time<10ms TTL=32
Reply from 204.96.64.171: bytes=32 time<10ms TTL=32
Reply from 204.96.64.171: bytes=32 time<10ms TTL=32
Reply from 204.96.64.171: bytes=32 time<10ms TTL=32
```

The following variables are allowed within a 'config' tag.

Errmsg
Timefmt
Sizefmt
Cmdecho
Cmdprefix
Cmdpostfix
Onerr

'odbc' tag

Attributes

The **odbc** tag provides for querying and updating odbc databases. Four variables are defined for the odbc token; 'debug', 'connect', 'statement', and 'format'.

Debug variable

The **debug** variable is used to set the SSI+ engine into debug mode. Debug mode provides diagnostics of a highly detailed nature from both the SSI engine itself and the local ODBC engine. Debug messages appear on the returned HTML document at the position at which the errors occur. The debug variable should be used only for development and must be removed before production. When the debug variable is present a warning message will appear indicating it's presence, this message is benign and will not affect any other aspect of the SSI+ engine. The format of the debug variable is :

```
'debug="'<debugstring>'" where;
```

<debugstring> is any string and is reserved for future use.

Connect variable

The **connect** variable is used to connect to a pre-existing odbc data source, to allow for subsequent **statement** tag operations on that data source. The format of the connect variable is

```
connect="'<datasource>', '<username>', '<password>'" where
```

<datasource> is the name of an odbc data source as defined on the local system in the odbc configuration utility. **CAUTION!** the account under which the server is run must be granted permission to access the data source.

<user name> is the name in which to log into the data source.

<password> is the password with which to access the data source.

Example. To connect to a data source called 'odbcsh1' as user 'foo' and password 'bar', one would use the following statement:

```
<!--#odbc connect="odbcsh1,foo,bar"-->.
```

Statement variable

The **statement** variable is used to submit a *Transact SQL* statement to the odbc data source. The format of a statement variable is: `statement="'<SQLStatement>'",`

where <SQL Statement> is any *Transact SQL* statement as defined in odbc and SQL reference text and help files.

Example. Suppose one wanted to query the 'CUSTOMERS' table from the above connected 'odbcsh1' database to return all rows and display each row on a separate line. One may use the following sequence of statements:

Connect to the database with a **connect** token as described above.

Setup the output format with a **statement** token as described below.

Execute the query: `<!--#odbc statement="SELECT NAME, AGE, VISCOSITY FROM CUSTOMERS ORDER BY 3, 2, 1" -->.`

Each row of the database will be inserted into the HTML page per the format statement as demonstrated below.

Format variable

The **format** variable is used to provide a template for the formatting of data returned from an `odbc` query. Use this variable to set up the appearance of data that will be returned from subsequent **statement** tag operations that return data from a database (i.e. the SQL statement 'SELECT').

The format of the format variable is

```
format="'<printfstatement>'"
```

where `<printfstatement>` is a standard C language `printf` format string with the restriction of only allowing string (`%s`) insertions. The user is referred to any C language text for a description of this format. The number of instances of `%s` must be equal to the number of fields selected in the subsequent SQL `SELECT` statement token.

Example. Suppose one wanted to query the 'CUSTOMERS' table from the above connected 'odbcsh1' database to display the columns 'name', 'age', and 'viscosity' with each row on a separate line. One may use the following sequence of statements:

Connect to the database with a **connect** token as described above.

Setup the output format: `<!--#odbc format="<P> The customer's name is %s, he is %s years old, and he prefers a motor oil with SPF %s viscosity" -->`.

Execute the query with a **statement** token as described above.

Each row of the database will then be inserted into the HTML page per the format statement. For example, if the database has 3 rows the HTML output would look something like this:

```
Customer's name is Conan, he is 29 years old, and he prefers a motor oil with SPF 15 viscosity
Customer's name is Kevin, he is 45 years old, and he prefers a motor oil with SPF 30 viscosity
Customer's name is Alan, he is 43 years old, and he prefers a motor oil with SPF 50 viscosity
```

The following variables are allowed within an 'odbc' tag.

Debug
Connect
Statement
Format

'email' tag

The **email** tag provides for sending an email whenever an HTML page is accessed or an HTML form is submitted. The nature of the variables below is defined in RFC 733. The variables 'fromhost', 'tohost', 'fromaddress' and 'toaddress' are required. All others are optional.

fromhost defines the name of the SMTP host sending the mail.

tohost defines the name of the SMTP host the mail will be sent to.

fromaddress defines the email address of the from party.

toaddress defines the email address of the recipient party.

message defines the message body to be sent.

subject defines the subject field of the message to be sent.

sender defines the email address sending party.

replyto defines the email address to which replies should be sent.

cc defines the courtesy copy email addresses.

inreplyto defines the inreplyto field of the message to be sent.

id defines the id field of the message to be sent.

Example. The following document sends an email. Suppose we have a form with datum : [First, Last, Middle Initial, Company, Address1, Address2, City, State, Zip, Country, Phone, Fax, Request, Urgency, ReplyMethod; Email, Subject, Message]. We may post that form to an HTML document containing the following fragment to send an email:

```
<!--#email fromhost=""www.theworld.com"tohost="mailbox.theworld.com" message="First
- &&First&&, Last - &&Last&&, MI- &&Middle Initial&&, Company - &&Company&&, Address
- &&Address1&&, &&Address2&&, &&City&&, &&State&&, &&Zip&&, &&Country&&, Phone -
&&Phone&&, Fax - &&Fax&&, Request &&Urgency &&via &&ReplyMethod&&, Message -
&&Message&& "fromaddress=""&&EMail&&"
toaddress="markw@mailbox.theworld.com"subject="WebQuest - &&Subject&&"
sender=""&&EMail&& "replyto=""&&EMail&&" cc="" "inreplyto="A WebQuest™ Automated E-
Mail" id="WebQuestEMail" -->
```

'if' tag

The **if** tag provides for conditional execution of SSI operations, and conditional printing of HTML text, based on logical comparisons. The format of the if tag is :

```
'if' '''<operand1>''' <operator> '''<operand2>'''<operation> where:
```

<operand1> is the first operand of a logical comparison statement.

<operand2> is the second operand of a logical comparison statement.

<operator> is the logical comparison method ['==', '!=', '<', '>', '!<', '!>'].

<operation> is the action to take if the logical comparison evaluates to TRUE [[goto](#), 'print', 'error', [break](#), 'errorbreak', 'printbreak'].

The operands may be any string or number (integer or floating point). In the event that both operands are numbers the comparison will be based on the value of the numbers. In the event that one or both of the operands are not numbers, the comparison will be based on the alphabetic order of the operands.

The special case of the NULL operand is defined by two quotes with no characters between them. The NULL operand may be used to check for the existence of form data from the remote client (see example below).

The operator defines what kind of comparison is performed on the operands:

'==' The equalto operator evaluates to TRUE if the operands are equal to each other.

'!=' The notequalto operator evaluates to TRUE if the operands are not equal to each other.

'<' The lessthan operator evaluates to TRUE if operand1 is less than operand2.

'>' The greaterthan operator evaluates to TRUE if operand1 is greater than operand2.

'!<' The notlessthan operator evaluates to TRUE if operand1 is not less than operand2.

'!>' The notgreaterthan operator evaluates to TRUE if operand1 is not greater than operand2.

'hasstring' The hasstring operator returns TRUE if the text string in operand2 is found in the operand1 string.

In the event that the logical comparison evaluates to FALSE, nothing happens. If the logical comparison evaluates to TRUE then one of the following operations may be performed:

[goto](#) causes a jump to a [label](#) token. The format of the [goto](#) tag is:

```
goto <label>
```

where <label> is the name of a label defined in a subsequent [label](#) tag.

'print' causes text to be printed. The format of the print tag is:

```
print <text>
```

where <text> is any HTML text or tag.

'error' causes the current [config](#).`error` message to be printed.

[break](#) causes termination of the HTML document transmission to the client.

'errorbreak' causes the current [config](#).`error` message to be printed, and then causes termination of the HTML document transmission to the client.

'printbreak' causes text to be printed, and then causes termination of the HTML document transmission to the client. The format of the printbreak tag is the same as the format of the print tag.

Example. The following document fragment compares two numbers. If the operands are not equal then a [goto](#) will jump to a [label](#).

```
<!--#if "10" != "20" goto testlabel -->
<P>This should not print
<!--#label = "testlabel " -->
<P>This should print
```

Example. The following document fragment demonstrates conditional execution based on data delivered from an HTML form. Suppose we have two form datum called 'formdata1' and 'formdata2' and we wish to compare them. The following document fragment compares the two operands. If the operands are equal then a [goto](#) will jump to a [label](#). Otherwise, the next line will print and the document will terminate on a [break](#) token.

```
<!--#if "&&formdata1&&" == "&&formdata2&&" goto testlabel -->
<P>The operands are not equal.
<!--#break -->
<!--#label = "testlabel " -->
<P>The operands are equal.
```

Example. The following document fragment prints two different statements depending on whether or not the client agent is NCSA Mosaic.

```
<!--#if "&&HTTP_USER_AGENT&&" hasstring "Mosaic" goto mosaiclabel -->
<P>You are not using Mosaic.
<!--#goto = "defaultlabel" -->
<!--#label = "mosaiclabel" -->
<P>You are using Mosaic.
<!--#label = "defaultlabel" -->
```

Example. Suppose we have a form with, amongst other things, a field named "BOO" and we wish to make sure that the remote client user entered data into the "BOO" field before submitting the form. The following document fragment checks for the presence of data in the "BOO" field. If data exists then nothing happens. If data does not exist then a message will be displayed and the document will terminate.

```
<!--#if "BOO" == "" printbreak "<P>You must provide data for the BOO field, please
resubmit." -->
```

'goto' tag

The **goto** tag provides for jumping to a label token without executing any SSI code or printing any HTML text between the goto token and the label token. The format of the goto tag is:

```
'goto ='<label>''
```

where <label> is the name of a label defined in a subsequent label tag.

Example. The following document fragment demonstrates a goto to a label.

```
<!--#goto ="testlabel" -->  
<P>This line will not print.  
<!--#label ="testlabel" -->  
<P>This line will print.
```

'label' tag

The **label** tag provides a place for a [goto](#) or [if..goto](#) token to jump.

The format of the label tag is:

```
label = "'<label>'"
```

where <label> is any string less than 51 characters long without any space (' ') characters.

When the SSI+ engine encounters a label token nothing happens. It is simply a place holder for a previous [goto](#) to jump to.

'break' tag

The **break** tag provides for termination of the HTML document at any point. When the SSI+ engine encounters a break token, the HTML document is immediately truncated and transmission to the client is ended.

Example. The following document fragment demonstrates a break token:

```
<P>This line will print.  
<!--#break -->  
<P>This line will not print because the document has been truncated and  
  transmission to the client is terminated.
```

