HyperText Markup Language

A web-page is created using an ASCII-file containing text and tags that define how the text should be treated. Traditionally this file has the extension **.html** or **.htm**. The HTML language is codified (this manual covers version 4.0), but technology develops faster and many browsers support features that are not a part of the standard (notably Explorer). If you use those features, your Web-page may look better, but if somebody uses a different browser, the outcome may be unpredictable.

The idea is simple. A browser reads the text from the source file and sorts it out. Some parts are interpreted as commands (tags, escape sequences, see below) and the rest is taken as the contents and arranged as specified by tags. Concerning the contents, with an exception noted further, the End-of-line character is treated as a space, more spaces are treated as one space. Obviously the way characters are printed also depends on coding (this applies to special characters that various languages use), we will get to it below. If the text uses only the good old ASCII, then there is no problem.

There are three characters that are not printed but have a special meaning:

"`<`" and "`>`" delimit tags;

"`&`" starts an escape sequence (which is always ended by semicolon `;`).

Actually, also the quote characters "`"`" and "`'`" have a special meaning, but this special meaning applies only to attributes within tags (see below). Thus when they appear within normal text (the content of the page), it is assumed that they should be taken literally and printed; pretty much all browsers are reasonable and do this, but formally it is not correct.

Escape sequences

Escape sequences help you print characters that are not available otherwise, for instance, by typing `<` you obtain `<`. General form is `&#`*number*`;`. Many popular escape sequences also have descriptive alteratives that use letters instead of numbers (note that they are case sensitive), for that `<` it is `<`. We need

`<`	`<`	<	`>`	`>`	>	`&`	`&`	&
`"`	`"`	"	`'`	`'`	'	` `	` `	

The first five escape sequences allow us to print the special characters. As noted above, if we use `"` directly in normal text, it will almost surely work, but we cannot do without `"` if we want to use double quotes within the argument of some attribute (see tags below). The same applies to single quote. Note that although all this is standard since the very beginning, practical implementation has been troublesome and using `"` and `'` can give unreliable results, it is better to use codes for these two.

The escape sequence ` ` is very useful, it is a non-breakable space. A line can't be broken there, moreover, they are not ignored or merged and thus can be used to create horizontal (and also vertical) blank spaces.

Note several features of escape sequences. First, the word descriptions are synonyms. As of HTML 4.0 there are 252 "character entity references" defined, see Appendix 2. Other symbols are defined just by numbers. Many escape sequences are just alternatives of the usual characters that we can type directly, in particular because code charts tend to be based on the ASCII table. Thus for instance `0` through `9` mean the characters 0 through 9, `A` through `Z` are upper-case letters A–Z and `a` through `z` are lower-case letters a–z. Similarly, `	` is a horizontal tab, `
` is a line feed and `
` is carriage return.

One has to be careful when using these codes, some are more reliable than others (the situation should improve with passing years, but even 10 years later some escape sequences are still not implemented by major browsers). There might also be trouble when other character coding is used (e.g. `iso-8859-2`) and still more trouble when you attempt to print them. For some insight and charts of escape sequences see Appendix 2.

Tags

Text is formatted on screen using tags. It is divided into units, each unit is treated in a certain way and often may be embedded into another.

A unit is started with an opening tag and (usually) ended with a closing tag. The opening tag consists of a text closed between `<` and `>`. The corresponding closing tag consists of the same text preceded by a slash `/`, enclosed in `<>` again. For example, if we want a part of text to form a paragraph, we start it with `<P>` and end it with `</P>`. The text following after `</P>` will start on a new line, most likely after some vertical space, perhaps indented. **Unless explicitly stated, all tags require a closing tag.**

Many opening tags also have optional attributes that modify the function of the tag. The format is generally `ATTRIBUTE=`*value*, *value* should be enclosed in either double quotes `"` or single quotes `'`; when we use one kind of quote, then we are free to use the other in *value*. One can also use quotes within *value* using escape sequences, see above.

If *value* consists only of letters, numerals, hyphens and commas, it need not be enclosed in quotes. Finally, note that HTML is not case sensitive, so for instance <TItle> and <tiTLE> will do the same. In this manual we will use capital letters to emphasise tags.

The tags that are paired (that have opening tags and closing tags, so actually almost all of them) create groups that cannot overlap, they must be nested. For instance, <P><B> *text* </B></P> is correct while <P><B> *text* </P></B> is wrong.

If you want to include comments in an HTML file, the format is <!-- *comment* -->. All parts (exclamation mark, hyphens) are necessary.

1. Basic Document

Every HTML-file should be one block called <HTML>. This in turn consists of two parts: The <HEAD> part identifies the Web-Page and the document itself is in the <BODY> part. So a minimal WWW-document may look like this:

```
<HTML>
<HEAD>
<TITLE>A Web-Page</TITLE>
</HEAD>
<BODY>
<P> First paragraph.  </P>
<P>Second paragraph.</P>
</BODY>
</HTML>
```

Actually, there should be one more tag before <HTML>. The tag <!DOCTYPE> is a standard way of specifying the type of document, the last expression identifies what type of HTML we are using. Options are:

```
<!DOCTYPE HTML PUBLIC "-//IETF//DTD HTML 2.0//EN">
<!DOCTYPE HTML PUBLIC "-//IETF//DTD HTML 3.2//EN">
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.0//EN">
```

Since this manual was written more advanced norms of HTML appeared (even XML), but they are backward compatible, so everything from this manual should work fine. People sometimes do not include this line and usually nothing bad happens, but with more and more mutually incompatible formats appearing every day (XHTML versus traditional HTML etc.), it is better to put it in.

1.1. Head

The <HEAD> section contains the following parts in arbitrary order:

• <TITLE> specifies the document title that is usually displayed on the top of the browser window and also identifies the document in a bookmark file. This is the only obligatory part of the head section, the rest is optional. Note that the following tags are not paired:

• The **non-paired** <META> tag supplies document information not defined by other elements. One uses NAME or HTTP-EQUIV attributes, in both cases CONTENT must follow. Some popular applications:

— specifying the character set to be used when displaying the page; some possibilities:

```
<META HTTP-EQUIV="Content-Type" CONTENT="text/html;charset=us-ascii">
<META HTTP-EQUIV="Content-Type" CONTENT="text/html;charset=iso-8859-2">
<META HTTP-EQUIV="Content-Type" CONTENT="text/html;charset=windows-1250">
```

— <META HTTP-EQUIV=refresh CONTENT="10;http://www.mysite/new.htm"> means that after 10 seconds, the page will be replaced by the one specified in CONTENT. With 0 seconds one can make automatic jumps to different pages, page can also refresh itself after a time. If we put, say, CONTENT="0; url=sounds/intro.wav", the sound will play when the page is loaded.

— <META HTTP-EQUIV=pragma CONTENT="no-cache"> tells the browser not to cache the page.

— <META HTTP-EQUIV=expires CONTENT="Fri, Apr 1997 17:46:01 GMT"> tells the browser when does the page expire, compatible browsers will look for a current page.

— <META HTTP-EQUIV=author CONTENT="Petr Habala">

— <META HTTP-EQUIV=generator CONTENT="QEdit v 2.0">

— tags <META HTTP-EQUIV=description CONTENT="a short terse overview of HTML language"> and also <META HTTP-EQUIV=keywords CONTENT="html,frames,tags"> help some search engines to classify the document.

• The **non-paired** <STYLE> tag specifies the style sheet for the page (see 5. CSS).

• The **non-paired** <BASE> tag specifies the base URL for relative links etc. using the HREF attribute (see 3.2. Links). Example: <BASE HREF="http://www.com/">. One can also specify a TARGET (window or a frame) to which all links are loaded unless overridden individually. See 4.1. Addressing Frames.

• The **non-paired** <LINK> tag serves as a means of referencing between separate documents. First you specify the relationship to the linked resource and then you specify its URL. For instance, <LINK REL=contents HREF=toc.html> tells the browser that the table of contents of the current document can be found in toc.html. Other similar values for the REL attribute are top (first page in a collection), index, glossary, copyright, next, previous, help, search.
Another example: <LINK REL=author HREF="mailto:me@address.com">.
This tag is now mainly used to link to a style sheet page, for instance
<LINK REL=stylesheet HREF="stylesheet2.css" TYPE="text/css">.
• The <SCRIPT> tag contains an in-line script that is usually hidden so that older browsers don't display it. The TYPE of scripting language must be specified, LANGUAGE is optional and serves no purpose. Example:

```
<SCRIPT TYPE="text/javascript" LANGUAGE="JavaScript">
<!--//hide the script here
document.write( "<br>Last updated "+ document.lastModified );
//end hiding the script -->
</SCRIPT>
```

Since this is not a Java manual, we will not go into it.

1.2. Body

The <BODY> tag should come right after </HEAD> and </BODY> should be the second last tag at the very end, just before </HTML>. <BODY> has the following optional attributes defining the look of the document.
• One can specify a picture as a background using BACKGROUND="*filename*". If the image is not large enough to cover the whole viewing window, it is automatically replicated and tiled.
• BGCOLOR describes the background colour, TEXT the text colour, LINK is for the colour of unvisited links, VLINK for the colour of visited links, and ALINK is used for links at the moment the user clicks on them. The value of these attributes is given by the RGB (red, green, blue) value, intensity of each colour is given in hexadecimal range 00–FF. For example, #9690CC is silvery and #00FF00 green. One can also use one of the following colour names (standard 16 colours of the Windows VGA palette):
black is #000000, silver is #C0C0C0, gray is #808080, white is #FFFFFF, maroon is #800000,
red is #FF0000, purple is #800080, fuchsia is #FF00FF (also called magenta), green is #008000,
lime is #00FF00, olive is #808000, yellow is #FFFF00, navy is #000080, blue is #0000FF,
teal is #008080, aqua is #00FFFF (also called cyan).
These sixteen colours are good because they work well on MacIntosh, Unix, and Windows machines. Because of different colour standards of these systems, more creative colours that work fine on, say, a Unix machine, are often not rendered well on, say, a Windows machine. Recenly, Netscape and Explorer have adopted a common standard of 216 colours, those that can be coded using 00, 33, 66, 99, CC, FF.

The <BODY> block contains elements that generally fall into two categories. Block level elements cause paragraph breaks, whereas text level elements are a part of the line where they were used. Block level elements can contain other block level or text level elements, but text level elements can contain only text level elements.

2. Block Level Elements

The paragraph tags <P>—</P> delimit **paragraphs**. There is usually some extra space between adjacent paragraphs. More consecutive <P> tags are treated as one. The opening tag <P> also closes the previous paragraph if there is any, so within HTML 4.0. the example above could have had the body

```
<P>First paragraph.
<P>Second paragraph.</P>
```

However, this may cause troubles (notably in XML that requires closing tags), so it is safer to always use </P>. The <P> tag has an optional attribute ALIGN with possible values left (default for <P>), center, right and justify. For instance,

```
<P ALIGN=center>Centred Text</P>
```

will be displayed as

Centred Text.

Note: The paired tag <CENTER>–</CENTER> can be also used to centre text, graphics, tables, ...

If we want to make a heading, we use a paired headings tag. There are six levels of headings, <H1> is the highest and <H6> the lowest. It is customary to start with <H1> and not to skip levels. Headings are displayed in larger and/or bolder fonts than normal body text. Example: <H1>Title</H1>. In opening tags we can use the ALIGN attribute with values as in <P>.
Many tag elements can be included in other constructions, but headings are not among them. Headings are supposed to stand apart, so for instance it wouldn't do to include them in lists or anchors. Although some browsers (e.g. Netscape) may handle such cases to your satisfaction, officialy the outcome is undefined.

This brings us to one important feature of the HTML language. Note that we don't specify how the headings will be formatted. This is typical, many things are specified logically and the implementation is up to the author of the browser. In other words, in headings you specify which of them are more important and which less, but the particular means (the size and type of the font) may differ from one browser to another. In fact, most of the HTML features work this way. This is just another reason for trying to follow HTML standards, because you may finetune your unusual document for a few browsers but never for all, including the future ones.

Another tool for aligning text is the `<DIV>` tag, which allows the `ALIGN` attribute and requires an end tag. This tag only splits the text into logical units and perhaps aligns it. Note that `<DIV>` will terminate the preceding `<P>` tag; however, there should be no paragraph break at this point. The real usefullness of this tag comes with style sheets, see 5. CSS.

2.1. Lists

HTML supports three kinds of lists. The format is similar and we can nest different lists.

Unnumbered Lists use bullets to denote items. Such a list is delimited with the paired `<UL>` tag and items are enclosed in paired `<LI>` tags. An item may contain more paragraphs. Example:

• bananas	can be obtained by	`<UL>`
• oranges		`<LI> bananas </LI>`
		`<LI> oranges </LI>`
		`</UL>`

The item mark can by changed by the `TYPE` attribute of `<UL>`, values can be `disc` (black bullet), `square` (black square), and `circle` (empty circle).

Similar are directory lists delimited by `<DIR>` and `</DIR>`. These also use `<LI>`, should contain only short items and cannot be nested. The same is true about `<MENU>`—`</MENU>` lists, many browsers render them more compact. These two kinds of lists are supposed to be discontinued.

Numbered Lists (or ordered lists) are done just like unnumbered lists, but we use `<OL>` tags instead of the `<UL>` tags. Items are numbered. The `<OL>` tag initializes the counting, the default is 1. We can initialize it with some other value using the `START` attribute in `<OL>` or set it with the `VALUE` attribute in a particular `<LI>`.

The `<OL>` tag also accepts the `TYPE` attribute controlling the numbering style. Values are `1` (arabic numerals), `a` (lower-case letters), `A` (upper-case letters), `i` (lower-case roman numbers), and `I` (upper-case roman numbers). Example:

iii. bananas	can be obtained by	`<OL TYPE=i START=3>`
iv. oranges		`<LI> bananas </LI>`
vii. apples		`<LI> oranges </LI>`
		`<LI VALUE=7> apples </LI>`
		`</OL>`

Definition Lists are delimited using `<DL>` and their format is a bit different. Each item consists of two parts. Paired `<DT>` contains a "definition term", the part that defines an item mark, and can only contain text level elements. The item itself (definition definition) is enclosed bay paired `<DD>` tags and can contain both text and block level elements except headings and the address element.

Example:

```
<DL>
<DT> Unnumbered Lists </DT>
<DD> Unnumbered lists precede ... bullets.  </DD>
<DT> Numbered Lists </DT>
<DD> Numbered lists are usually ... several levels of numbering.  </DD>
</DL>
```

will do

Unnumbered Lists
: Unnumbered lists precede items with all kinds of marks, usually full bullets and empty bullets.

Numbered Lists
: Numbered lists are usually numbered using arabic numerals. HTML browsers should support several levels of numbering.

If your definition terms (the `<DT>` parts) are short, you may ask the browser to fit them next to the items, this is done using the optional attribute `COMPACT` in `<DL>` (it is a flag, that is, it has no value assigned). This attribute can be actually used with all types of lists, suggesting a more compact style.

If you want to put several lists as a part of one item (<LI>, <DT> or <DD> part), it is recommended to enclose each list into a paragraph. Example of a nested list:

```
<OL>
<LI> Parameters of <TT>copy</TT>:
<P><UL>
<LI> file to be copied</LI>
<LI> target directory</LI>
</UL></P></LI>
<LI> Options of <TT>copy</TT>:
<P><DL COMPACT>
<DT> /A</DT>
<DD> (ASCII) File is copied up to but not including the first Ctrl-Z character.</DD>
<DT> /B</DT>
<DD> (binary) The whole file is copied.</DD>
</DL></P></LI>
</OL>
```

This will be perhaps displayed as

1. Parameters of `copy`:
   - file to be copied
   - target directory
2. Options of `copy`:
   /A (ASCII) File is copied up to but not including the first Ctrl-Z character.
   /B (binary) The whole file is copied.

2.2. Tables

The whole table is delimited by the paired <TABLE> tag. Possible attributes: ALIGN (values as in <P>, default left), WIDTH (in pixels WIDTH=110 or as percentage WIDTH="70%"), CELLSPACING (distance between cells of the table), CELLPADDING (distance between the border and contents of cells), and BORDER. If BORDER=n is present, elements of the table are separated by borders of specified width in pixels, otherwise they are adjacent seamlessly (this way one can for instance make a huge picture out of several small ones).

The first item inside a table is optional, it is the tag <CAPTION>. The text between <CAPTION> and </CAPTION> is by default centred at the top of the table (ALIGN=top), it can be centred below the table if the attribute ALIGN=bottom is present. Only text-level elements are alowed inside. We can also use values left, right, and center for ALIGN, but then we have to use VALIGN for top or bottom.

Then the contents of the table is described, row by row. Each row is enclosed by <TR> and </TR> (which can be left out but we do not recommend it) and is formatted independently of others. Entries in each row are treated using certain defaults. These defaults can be changed in a particular row by optional attributes: ALIGN for horizontal alignment of text in entries (options are left, center, right); and VALIGN for vertical alignment of entries (options are top, middle, bottom, baseline).

Entries within a row can be specified using the pairs <TH>–</TH> and <TD>–</TD> (closing tags can be left out but again, don't do it). The former is great for making table headers, because the text is centered in boldface. The latter is better for regular entries, the text is aligned left and centred vertically. Both <TH> and <TD> may contain attributes that override defaults or attributes from the corresponding <TR>. One can use the same attributes as <TR> has, and some extra ones: COLSPAN=n means that the entry will span n columns (default 1), ROWSPAN=n means that the entry spans n rows (default 1), and NOWRAP turns off word wrapping within an entry. WIDTH and HEIGHT suggest specified sizes for the cell (in pixels).

Note: If we specify that some entry spans several rows, then it extends downward into the next rows, therefore when specifying the contents of those subsequent rows we don't specify anything for that particular column. Example:

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

will do

<table>
<tr><td rowspan="2">cell 1</td><td>cell 2</td></tr>
<tr><td rowspan="2">cell 3</td></tr>
<tr><td rowspan="2">cell 4</td></tr>
<tr><td>cell 5</td></tr>
</table>

We can also format whole columns. This is done using the paired tag <COLGROUP> right after the <TABLE> tag. We can use several such groups to cover all columns. We have two choices:
— If the pattern is the same for several successive columns, we can specify it as attributes of <COLGROUP>, namely ALIGN (for text in columns, values left, right, center, justify), VALIGN (text alignment, values top, bottom, middle, baseline), WIDTH (the width of the column in pixels), and SPAN (how many columns are acted upon.)
— Several different patterns can be specified within one <COLGROUP> group by using a (non-paired) tag <COL> for each column, attributes are the same. Example:

```
<TABLE BORDER=1>
<COLGROUP>
<COL VALIGN=top ALIGN=left>
<COL VALIGN=top ALIGN=center>
</COLGROUP>
<COLGROUP VALIGN=middle ALIGN=center SPAN=3></COLGROUP>
```

and then we put the contents of the table as usual.

Finally, we can specify colours. This only works if the BORDER attribute is present in <TABLE>. The attributes BGCOLOR and BORDERCOLOR can be used in any of <TABLE>, <TR>, <TH>, <TD> and as usual, in every tag a specification overrides the ones in tags before it in this list. Text can be coloured using <FONT> or <BODY>. Note: To change fonts, <FONT> has to be used in each cell.
<TABLE> and <TD> tags can also have the BACKGROUND attribute (see <BODY>), the picture is then cropped or tiled as needed. This is not standard and each browser does it differently.

One can use <TABLE> to put text of the page in columns. For instance, to get two columns, one would use

```
<TABLE WIDTH=100%>
<TR>
<TD WIDTH=50%> Left hand column text in here. </TD>
<TD WIDTH=50%> Right hand column text in here. </TD>
</TR>
</TABLE>
```

Note: Using   we can create an empty cell.

2.3. Other Formatting Features

The paired <PRE> tag designates a preformatted text. Text in a <PRE> section is interpreted literally, prints as is including linebreaks and empty lines. Hyperlinks can be used withing <PRE> sections, but one should avoid other tags. Note that special HTML characters <, >, and & can be printed only using their escape characters (see above) even in preformatted text. <PRE> has an optional WIDTH attribute.
Similarly works <XMP>–</XMP> but differs in one crucial aspects: the tags inside are not interpretted but shown literally.
The <BLOCKQUOTE> tag is used for lengthy quotations and browsers usually change margins to separate it visually from surrounding text.
The <ADDRESS> tag usually appears at the end of a file and contains an address in form of plain text, perhaps with paragraphs and text-level elements. Many browsers like to change font to highlight the address.

Now come two handy commands for direct formatting. We know that <P> breaks lines, but it may leave an extra space between lines. If you only want to break a line, use the command
. This one is not paired, it simply breaks a line. It has an optional attribute CLEAR with values left, right, or all. It tells the browser to start the next line lower in case there is some in-line graphics so that the text can start with a clear left margin (or right or both). We can insert soft line breaks with <WBR> (non-paired).
The paired <NOBR>–</NOBR> tag tells the browser that the text inside can not be broken. If the text is too long, a horizontal scroll should appear. Note that this highly useful tag is not a part of the official specification, but most browsers recognize it, which is good as there is no alternative.
<HR> is not paired. It draws a horizontal line, implicitly the width of the browser window. It has four optional attributes. The line is centred by default, we can change it using the ALIGN attribute with values left, right, center. If we use the flag NOSHADE, the line is rendered in full colour. The thickness (in pixels) is controled by SIZE, say, SIZE=4. The optional argument WIDTH determines the width of the line as percentage of the browser window width (say, WIDTH="50%") or in pixels (say, WIDTH=100).
<HR NOSHADE SIZE=4 WIDTH="50%"> could look like this:

There is no vertical line tag. One way to create some is to use tables. For instance, to create a vertical line to the left of our text (for which an image of a line would not be practical, you never know how large your

text will be on a certain browser), we use the following:

```
<TABLE WIDTH="100%" CELLPADDING=0>
<TR><TD WIDTH=2 BGCOLOR="black">&#124;</TD>
<TD><P> Text here
</P></TD></TR>
</TABLE>
```

Some browsers don't like empty cells, so we put the vertical rule there as it is not too wide.

The paired `<FORM>` tag is used to define an HTML form. The `<FORM>` block may contain a wide range of HTML elements, including single- and multi-line text fields, radio button groups, checkboxes, and menus. It makes little sense going into it without discussion of Pearl or Java etc., so to have clear conscience we decided to show the HTML side of it in the Appendix 1.

3. Text Level Tags

These don't cause paragraph breaks.

3.1. Fonts

There are two kinds of font tags, logical and physical. Physical tags tell browsers directly what font must be used. Logical tags tell the browser what you are talking about and it is up to the browser to decide how to write it. Of course, all font tags have necessary closing tags. Logical tags:
`<DFN>` is for definitions, typically displayed in italic,
`<EM>` is for emphasis, typically displayed in italic,
`<CITE>` is for titles of books etc., typically displayed in italic,
`<CODE>` is for computer code, typically displayed in typewriter,
`<KBD>` is for user keyboard entry, typically displayed in plain fixed-width font,
`<SAMP>` is for sequences of literar characters, typically displayed in typewriter font,
`<STRONG>` is for strong emphasis, typically displayed in boldface,
`<VAR>` is for a variable, typically displayed in italic.

Physical tags: `<B>` bold, `<I>` italic, `<TT>` typewriter, `<U>` underlined text, `<S>` or `<STRIKE>` puts a horizontal line throught the text, `<BIG>` large font, `<SMALL>` small font, `<SUB>` subscript, `<SUP>` superscript. Note: If `<BIG>` is used around `<SUB>` or `<SUP>`, then the sub/superscript will be in the usual text size.

If you use several tags in a row, the outcome is undefined. Some browsers take only the last that was used, some "add" them, e.g. `<B><I>` would give boldface italic.

Netscape has a `<BLINK>` tag that makes the text blink.

There is also a `<FONT>` tag, which requires `</FONT>` and the text between these two tags is rendered in size specified by the `SIZE` attribute and colour specified by the `COLOR` attribute of `<FONT>`. The third optional atrribute `FACE` contains a comma-separated list of fonts that the browser can use in order of our preference, e.g. `<FONT SIZE=3 COLOR=blue FACE="arial,courier,gills sans">`.

The size can be set to an integer from 1 to 7 for an absolute size font. It can be also specified relatively, but then we first have to define the base font size by the `SIZE` attribute of `<BASEFONT>` (which does not do anything else than set the base size). After this, when we use the `SIZE` attribute of `<FONT>`, its value is added to the base font size, the sum should be between 1 and 7. Actually, `<BASEFONT>` has two more possible attributes, `COLOR` and `FONT`.

Note that the `<FONT>` tag goes against the basic principles of HTML, since it is "hard", while specification in `<BODY>` is "soft". This may cause troubles. Example: You set your background and text in `<BODY>` to show white on black, at one place you emphasise using yellow in `<FONT>`. Somebody is somewhat colour-blind and sets up his browser to show everything blue on yellow. On your page this will overrule the "correct" `<BODY>` assignment, but not the `<FONT>` specification and the emphasised text will be—yellow on yellow, invisible.

3.2. Links

A hypertext link is a piece of text or an image that serves as a pointer to another document. Links are done using anchors determined by the paired tag `<A>`. It has an obligatory attribute `HREF="`*link*`"`. The text or the picture between `<A>` and `</A>` is used to identify the link. It is somehow highlighted by a browser and clicking on it calls the *link*.

One can link to another HTML file on the same machine. This is done using the UNIX syntax. It can be either a relative pathname or an absolute pathname. A relative pathname starts from the current path. Say, if your document is in `/user/myfiles` and you put an anchor `<A HREF="../myotherfiles/one.html">File one</A>` there, then a browser will display and highlight File one and when you click on it, it will download the file `/user/myotherfiles/one.html`.

Absolute pathnames are links that specify a complete URL.

URL (Uniform Resource Locator) is a universal way of specifying resources on Internet. The syntax is *scheme://host.domain:port/path/filename*,
where the *:port* part is usually not necessary. The type of document is specified by *scheme*, possibilities are
`file` for a file on the local system,
`ftp` for a file on an anonymous FTP server,
`http` for a file on a World Wide Web server,
`gopher` for a file on a Gopher server,
`wais` for a file on a WAIS server,
`telnet` for a connection to a Telnet-based service,
`news` for a newsgroup,
`mailto` is for e-mailing, here the syntax is `mailto:`*e-mail@address*. One can put more addresses separated by commas.

There are different kinds of files you can refer to. Browsers check on their extensions and call appropriate software to display these files. Some common MIME types are: plain text (`.txt`), HTML document (`.html` or `.htm`), PostScript file (`.ps`), image (`.gif` or `.jpeg` or `.jpg` or `.tiff` or `.xbm` (XBitmap)), sound (`.aiff` or `.au` or `.wav`), movie (`.mpeg` or `.mpg` or `.mov` (QuickTime)).

When referring to documents, one can actually refer to specific places. First, in a file you refer to, you place a "named archor", using the tag `<A>` with an attribute `NAME="`*label*`"`. Then, when referring to the document, you attach `#`*label* after the filename. For instance, if I want to refer to the second section of file `myfile.htm`, I would put `<A NAME="sec2"></A>` at the beginning of the second section, and then in the current file I would put an anchor `<A HREF="myfile.htm#sec2">Go there</A>`.

When referring to a specific section of the current document, it is not necessary to specify the file in an anchor, it is enough to write `HREF="#`*label*`"`.

We should mention three more attributes of `<A>`. `TARGET` specifies to which frame/window the link should be loaded. If such a window does not exist, it is created so that the user suddenly has an extra browser window and the original document is still visible (see 4.1 Addressing Frames). `REL` and `REV` are not widely used, can be used only with the `HREF` attribute and specify the relationship of the link to the current page (the former) or the reverse (the latter).

3.3. Embedded Images

Images can be a part of an HTML document. They are embedded using a **non-paired** `<IMG>` tag. The image is specified similarly to links (i.e. the whole URL or just a relative path if it is on the same machine) using the attribute `SRC="`*image.name*`"`. Filename must have a proper extension, see above. The size of the image should also be specified in pixels using attributes `HEIGHT` and `WIDTH`, it speeds up loading. For instance, `<IMG SRC="pic.jpg" HEIGHT=100 WIDTH=65>` would embed the picture `pic.jpg` into the document.

Note that if you don't give actual dimensions of the picture, some browsers will shrink/enlarge it to fit the specified size, others will not. Some browsers figure out the size when not specified.

Images may be a part of an anchor; this is the only tag which is recommended to use inside an anchor. For instance, to save downloading time, you may decide to show only a small picture in your document, with a large one available by clicking on the small one. This would be done using the link

```
<A HREF="largepic.jpg"><IMG SRC="smallpic.jpg"></A>
```

Note that some browsers cannot display images, or are configured not to. In that case it is possible (and courteous) to specify a text that appears instead of the image. This is done using the optional attribute `ALT="`*text*`"`. It has another application: the text is shown when the cursor is positioned over the picture. So if we use the picture ⇐ as an anchor for going back with `ALT=Back`, the user will appreciate that the word *Back* pops up when he points at that strange arrow.

There are two important positions for pictures. One possibility is to put the picture into a paragraph of its own and use the paragraph attribute `ALIGN` to `center` the picture or put it `left` or `right`.

We can also place the `<IMG>` tag right into a piece of text. The picture is then aligned by its bottom with the current line and after the next line break, the text continues below the picture. The alignment with respect to text can be changed using the attribute `ALIGN`. Default is `bottom` or `baseline`, the values `top` and `middle` change the alignment accordingly. When these values are used, then text still continues after a line break under the picture. Unfortunately, the `middle` value is not defined properly, so some browsers align middle level of the picture with the middle of the text, while others align middle of the picture to bottom of the text. Better defined are values `texttop` (aligned with the top of the text), `absmiddle` (uses the middle of the current line), and `absbottom` (aligned with the bottom of the current line), but these are not universally supported.

If we use the values `left` or `right`, the picture is put at the nearest available place at the specified margin and the text wraps around it nicely, after a line-break it continues next to the picture if there is enough room. There is no provision to put a picture in the middle of the line and having the text flow around it left and right.

Other attributes are BORDER, which sets the width of the picture border in pixels (BORDER=0 suppresses the border) when used as a hypertext link, HSPACE and VSPACE define the amount of white space around the image in pixels. Newer browsers support LOWSRC. The image specified by this URL is loaded first, then the contents of the page, and then the image specified by SRC itself. This allows you to offer a low-resolution "preview" that loads fast, the real picture then fades in.

Sometimes an image is used to call different links depending on where we click on it. This can be done as a client-side image map, when the decision about links is handled by the browser, or as a server-side alternative, when the decision is made by a processing script on a server, which is much slower but also works for older browsers. In each case we load an image using the <IMG> tag and use a special attribute.

The server-side image map is created by using the attribute flag ISMAP. When the image is clicked on, the location is attached to the picture's URL and the processing script is called.

Example: <A HREF="bar.map"><IMG SRC=bar.jpg ISMAP BORDER=0></A>. If the user clicks on (10,27), the browser will call bar.map&10,27.

The client-side image map is created by defining the clickable areas of the picture, the so-called URL fragment identifier. This is done by the paired <MAP> element with some *name*. First, the <IMG> tag has to contain the USEMAP="#*name*" attribute. Then the <MAP> tag comes with NAME=*name* (the same as above of course). Between <MAP> and </MAP>, description of clickable areas follows using the non-paired <AREA> tags.

Each <AREA> contains the following attributes: HREF specifies link accessed when the defined area is clicked. ALT can be used for alternative text. The pair SHAPE and COORD defines the clickable area. Possibilities are:
— SHAPE=rect COORD="*left-x,top-y,right-x,bottom-y*" specifies the upper left and lower right corners of a rectangular area.
— SHAPE=circle COORD="*center-x,center-y,radius*" specifies the center and radius of a circular area.
— SHAPE=poly COORD="*x_1,y_1,x_2,y_2, ...* " specifies a polygonal area.
— SHAPE=default specifies the remaining undefined area of the image.

One can also cut a hole in other areas using the AREA tag with the NOHREF attribute flag. When more maps overlap, the first defined takes preference.

The coordinates are counted from the top-left corner of the picture and can be given either in pixels or in percentages of the picture's size.

The TARGET attribute can also be used to specify where to load the link (see 4.1. Addressing Frames).

4. Frames

Frames allow a web page to be split into several units, each of them behaving like a separate web-page. The basic document of the framed page has the following form:

```
<HTML>
<HEAD>
</HEAD>
<FRAMESET>
</FRAMESET>
</HTML>
```

The <FRAMESET> command splits the page into frames. Frames can be horizontal (using the attribute ROWS) or vertical (the attribute COLS). The value of these attributes is a list of height/width values.

In the list of values we can include any or all of the folowing types of specification:
— a number specifies height/width of a frame in pixels,
— a number followed by the % sign specifies height/width as a portion of the total space available. If only the above two kinds are used and total of percentages does not add up to 100%, requirements are scaled to give 100%.
— a number followed by an asterisk *. The '*' character represents a "relative-sized" frame. After assigning frames of lengths specified using pixel/percentage size, the remaining part is split among "relative-size" specifications in given ratios. For instance, a specification '2*' gets twice as much as a '*' specification.

Example: <FRAMESET ROWS="50,15%,2*,*,3*,10%"> will split the window into 6 horizontal parts. The first one has height of 50 pixels. The second and last have height of 15, resp. 10 percent of the total window. The third, fourth and fifth window split the remaining height in the ratio 2:1:3.

Between <FRAMESET> and </FRAMESET> we specify the contents of each frame in the order they were defined in <FRAMESET>. Frames are specified using the tag <FRAME> and the contents of the particular frame is determined by a link using the SRC attribute.

For instance (showing also the features described below),

```
<HTML>
<FRAMESET COLS="20%,80%">
<NOFRAMES><BODY>
If you had frames-capable browser, you would see the main menu.  Bummer.
</BODY></NOFRAMES>
<FRAME SRC="index.htm">
<FRAME NAME=main SRC="intro.htm">
</FRAMESET>
</HTML>
```

This will create a web page split into two vertical frames. The left fifth of the window contains a frame with a clickable index, specified as the web-page `index.htm`. The right window will be used to show the page requested by clicking at appropriate place in the left window, in the beginning we put there an introductory page `intro.htm`

The following optional attributes can be used with `<FRAME>`:
— `SRC` specifies the contents of the frame. If not used, an empty frame is created.
— `NAME` assigns a name to the frame so that it can be targeted.
— `MARGINWIDTH` can specify the width of the left/right margin of this particular frame.
— `MARGINHEIGHT` can specify the height of the upper/lower margin of this particular frame.
— `SCROLLING` has values `yes`, `no`, or `auto` and determines whether this frame should have a scrollbar. Default value is `auto`.
— `NORESIZE` is a flag, this frame will not resizable by a user. By default, all frames are resizable.
— `FRAMEBORDER` with values `yes`, `no` (for Netscape, `0` for Explorer) can be also used in `<FRAMESET>` and is not standard. To join frames seamlessly in both Netscape and Explorer, one needs to use `FRAMEBORDER=0 BORDER=0 FRAMESPACING=0` with all `FRAMESET`'s.

The above example also shows another possible tag used within the `<FRAMESET>` construction. The contents of `<NOFRAMES>`–`</NOFRAMES>` provides an alternative for browsers that are not frames-capable.

One can embed frames within frames by using another `<FRAMESET>`–`</FRAMESET>` pair instead of some `<FRAME>`. This corresponding frame will then be divided according to the embedded specification. For instance, this divides the main window into parts corresponding to the example in 2.2. Tables:

```
<FRAMESET COLS="50%,50%">
   <FRAMESET ROWS="50%,50%">
     <FRAME SRC="cell1.htm">
     <FRAME SRC="cell4.htm">
   </FRAMESET>
   <FRAMESET ROWS="25%,50%,25%">
     <FRAME SRC="cell2.htm">
     <FRAME SRC="cell3.htm">
     <FRAME SRC="cell5.htm">
   </FRAMESET>
</FRAMESET>
```

We could have also used `<FRAMESET ROWS="*,*">`, resp. `<FRAMESET ROWS="*,2*,*">`.

In fact, the pages we refer to in `<FRAME>` can themselves define more frames (that is, be of `<FRAMESET>` type). This way one can create a page where by clicking on one link, several frames are replaced - they are all part of another frameset referered to by the link and targeted to a `<FRAME>` of the parent document.

Now a question: How do I prevent my document from becoming a frame in somebody's page? Using Java. There are two cases:
— My document does not use frames. Then we add an attribute into the `<BODY>` tag:

```
<BODY onLoad="if (self != top) top.location = self.location">
```

— If my document is in a framed page, it gets interesting. We have to check whether it is in the right frame and if not, load our frameset. Let's say that our frameset is in `http://site.com/main.htm` and the document should be loaded into our frame named `mainarea`. Then we have to include the following in the `<HEAD>` area of the document:

```
<SCRIPT LANGUAGE="JavaScript">
<!--//
if (self != top) if ( top.frames[1].name !="mainarea") top.location =
"http://site.com/main.htm"
//-->
</SCRIPT>
```

It is possible and better to use `top.location = self.location`.

4.1. Addressing Frames

This is actually more general. Any window or frame can be assigned a *name*. Frames are assigned names using the NAME attribute (see above). A named window is created in two ways. A document can be loaded with `Window-target:` *name* in an optional HTTP header, or the link that called this document can have a TARGET attribute (see e.g. <A>). In both cases, if a window of that *name* exists, the document is loaded there. If such a window does not exist, it is created and given the *name*. Note that a window *name* cannot start with the underscore _ character.

Anyway, we have a frame or a window with a *name*. We want documents clicked on in another window/frame to load to the window/frame with the *name*. We do it using the attribute TARGET=*name* in one of the following places:

— when used in the <BASE> tag, all links in the document will be targeted to that window/frame unless overridden by the link itself. This is for instance useful in the example above with menu in the left frame and particular choice appearing in the right frame. We would put <BASE TARGET=main> to the file index.htm.
— when used in the <A> or <AREA> tag, this particular link will be loaded to the specified window/frame.
— it can be also used with the <FORM> tag, the result of the submission is then displayed in the specified window/frame.

User names of windows/frames are not allowed to start with the underscore character because there are some special reserved names that start this way:

— _blank creates a new window without a name to load the link.
— _self loads the link to the same window/frame; this is used to override global <BASE> targets.
— _parent loads in the immediate <FRAMESET> parent of this document (thus ignoring all sub-frames). If no parent is present, it acts as _self.
— _top loads into the top window, into its full body. This is a good way to escape from deep frame nestings. Also, if you are in a frame and call a link, it loads by default into this frame, which is not nice if you for instance refer to a separate Web-page, so here again TARGET=_top helps.

5. Cascading Style Sheets

Important note: If you are using sheets, you *must* close all tags although it is not normally necessary (like </P>, </LI>, </TD>, </OPTION>, ...).

This is not in fact a part of HTML but an extension. CSS provides definitions that apply to one or more HTML elements (for instance it is a way to conveniently set attributes for all <P> at once rather then to type them individually at all places). It is within intentions of HTML, that is, the preferences of CSS are then compared with browser's intentions and a compromise (precisely defined by CSS definition) is reached.

A style specification has the following format: `selector {property:value}` More `property:value` pairs for the same selector can be used, separated by semicolons. Example:

```
BODY {background:  #FFFFD8;
     margin-top:  20}
A:link {color:  #400080;
     background:  #FFFFD8}
H1, H2 {font-weight:  bold;
     text-align:  center;
     color:  blue;
     background:  #FFFFD8;
     font-family:  'Gill Sans', Arial, sans-serif}
CITE {font-family:  'Gill Sans', Arial, 'New Times Roman';
     font-style:  italic}
```

Note how the third definition applies to two tags. The definition of <A> only applies to links, for other options see below.

A very powerful weapon is a "class specification", allowing more styles for one tag. Here:

```
P.question {color:  green;
     font-style:  italic}
.answer {color:  red}
```

we define two classes, the first one just for paragraphs, the second one can be applied to any element. Now we can use <P CLASS="question"> and <P CLASS="answer"> to have paragraphs of two colours. Class names cannot start with a dash or a digit.

The ID definition is similar to the class definition without an attached HTML tag. For instance:

```
#mystyle {font-style:  italic;
     font-size:  smaller}
```

allows us to use <H1 ID="mystyle">. The ID name must be an initial letter followed by letters, digits or

hyphens and can be associated with one and only one element per document (so it is better to avoid ID). However, it is useful in dynamic HTML.

These definitions can be specified in three ways:
— If they are common to several documents, it is best to put them into a file with extension `.css`. There cannot be anything else (like tags) in such a file, but we can add comments enclosed between `/*` and `*/`. Documents that use it will link to it using `<LINK REL="stylesheet" HREF="`*filename*`.css" TYPE="text/css">` in the `<HEAD>` part. One can combine multiple sheets by using several `<LINK>` statements with the same `TITLE` attribute.
— One can embedd them to the current document to which they should apply. This happens in the `<HEAD>` part and the format is as follows:

```
<STYLE TYPE="text/css">
<!--
style definitions
-->
</STYLE>
```

Note that the definitions are a comment, so browsers not supporting styles will not show them. One can also import external style sheets this way. In the situation that we have a "master style sheet" and some particular sheets, we can link (see above) the main one and then, in the `<STYLE>` part, import more files that have lower priority than the linked sheet. These import statements must come before all other (style) statements in `<STYLE>`. Example:

```
<STYLE TYPE="text/css">
<!--
@import url(http://www.site.partial1.css)
@import url(http://www.site.partial2.css)
other style definitions
-->
</STYLE>
```

The sheet imported last overrides the one before it etc.
— Styles can be also applied inline to groups or individual tags. Every tag may contain the attribute `STYLE` whose value is a semi-colon separated list of property definitions. Example:

```
<P STYLE="background-color:#F6F6FA; color:#860BA8">
Wow, this <B STYLE="font-size:120%">is</B> something!</P>
```

We can also apply certain definitions only to some parts of the document. For this there are two paired tags that have no meaning, they only serve as tools of setting up styles. If a browser does not support styles, they are meaningless, otherwise the attribute `STYLE` is in effect. `<SPAN>` is a text-level tag, `<DIV>` is a block-level tag.

One can also use "contextual selectors". For instance the definition
`H1 STRONG { background: yellow }`
only applies if `<STRONG>` is used within some `<H1>` group. We can also make several definitions this way, say, if we use
`TD P CODE, H1 EM {color: red}`,
the specification will apply to `<CODE>` if it appears within `<TD>` `<P>`, and to `<EM>` in case it appears within `<H1>`.

One can thus have many definitions. Their interaction is defined by rules of cascading order. For each particular `property`, the most specific is used (that is, properties are inherited individually from more general to more specific, until/unless overridden). Linked definitions are least specific, inline styles most specific. Also, contextual selectors are more specific then general selectors (in the given context). For instance, if `H1 {color:red; size=12 pt}` is in a linked file and `H1 {color:blue}` is embedded, then the document will have `<H1>` blue of size 12pt.
One can influence the cascading order by adding the word `! important` after a `property` declaration, such definition then overrides all others. If both author's and reader's statements are done this way, the author's specification will win. Example:
`BODY {background: white ! important; color: black}`.

Here is a list of all currently supported properties that can be defined. Note that *lenght* can have many kinds of units, like `pt`, `px`, `em`, `in`, `mm`, ... :

• *Fonts* (can be applied to all HTML elements and are inherited):
`font-family:` list of names. It is recommended to also supply some generic names in case the browser does not have the ones you like. Valid generic names are `serif`, `sans-serif`, `fantasy`, `cursive`, and `monospace`;
`font-style: normal` or `italic`;

`font-variant:` `normal` or `small-caps`;
`font-weight:` `normal` or `bold`;
`font-size:` *percentage* or *size* in points;
`font:` combined specifications, e.g. `bold 12pt Arial`.

• *Colour and Background* (apply to all elements except where noted, not inherited unless noted):
`color:` *specification* (name or code), is inherited;
`background-color:` *specification*;
`background-image:` `none` or *URL* in the form `background-image:url(`*filename*`)`;
`background-repeat:` `no-repeat`, `repeat`, `repeat-x`, `repeat-y` (controls tiling of the image);
`background-attachment:` `scroll` or `fixed` (whether the background scrolls with the window);
`background-position:` `top` or `bottom` or `center`, `left` or `right` or `center`, for example `background-position: top center`, applies only to block level and replaced elements;
`background:` combined specification, e.g. `background: url(bgRock.jpg) repeat-y fixed top`.

• *Text* (inherited unless noted otherwise):
`letter-spacing:` `normal` or *length*, applies to all elements;
`vertical align:` `sub` or `super`, applies to inline elements, not inherited;
`line-height:` `normal` or *length* or *percentage*, applies to all elements;
`text-decoration:` `none`, `underline`, `overline`, `line-through`, applies to all elements, not inherited;
`text-transform:` `capitalize`, `uppercase`, `lowercase`, or `none`, applies to all elements;
`text-align:` `left`, `right`, `center`, or `justify`, applies to block-level elements;
`text-indent:` *length* or *percentage*.

• *Boxes* (not inherited, apply to block-level and replaced elements unless noted):
`margin:` `auto`, *percentage*, or *length*. One can specify all sides by listing for values separated by spaces (`margin: 10px 5px 10px 5px`). Or, each side can be specified as a separate property, that is, we have properties `margin-top:`, `margin-right:`, `margin-bottom:`, `margin-left:`;
`padding:` *length* or *percentage*, works just like `margin` including the possibility of four values and variants like `padding-top`;
`border-width:` `thin`, `medium`, `thick`, or *length*, works just like `margin`, including the possibility of four values and variants like `border-top-width`;
`border color:` *specification*, works just like `margin`, including the possibility of four values (`border-color: green #2F3056 white blue`) and variants like `border-top-color`;
`border-style:` `none`, `solid`, `double`, `groove`, `ridge`, `inset`, `outset`, works just like `margin`, including the possibility of four values and variants like `border-top-style`;
`border:` combined specification for the whole border, e.g. `border: thin solid blue`, there are four specific properties `border-top` through `border-left`;
`float:` `none`, `left`, or `right`, applies to `<DIV>`, `<SPAN>`, and replaced elements;
`clear:` `none`, `left`, `right`, or `both`, applies to all elements.

• *Positioning* (not inherited):
`clip:` `auto` or *shape* (`clip:rect(0px 200px 200px 0px)`), applies to all elements;
`visibility:` `visible`, `hidden`, or `inherit`, applies to all elements;
`position:` `absolute`, `relative`, `static`, applies to all elements;
`overflow:` `visible`, `hidden`, `scroll`, or `auto`, applies to all elements;
`height:` `auto` or *length*, applies to `<DIV>`, `<SPAN>` and replaced elements;
`width:` `auto`, *length*, or *percentage*, applies to `<DIV>`, `<SPAN>` and replaced elements;
`left:` `auto`, *length*, or *percentage*, applies to absolutely and relatively positioned elements;
`top:` `auto`, *length*, or *percentage*, applies to absolutely and relatively positioned elements;
`z-index:` `auto` or *integer*, applies to absolutely and relatively positioned elements.

• *Lists* applies to list item elements, inherited:
`list-style-type:` `disk`, `circle`, `square`, `decimal`, `lower-roman`, `upper-roman`, `lower-alpha`, `upper-alpha`, `none`;
`list-style-image:` `none` or *URL* (see background), replaces the item mark;
`list-style-position:` `inside` or `outside`;
`list-style:` combined specification, e.g. `list-style: square outside url(icFolder.gif)`.

and some others:
`display:` `none`, `block`, `inline`, or `list-item`, indicates when things are to be displayed, not inherited, applies to `<TABLE>`, `<INPUT>`, `<TEXTAREA>`, `<DIV>`, `<SPAN>`, `<IMG>`, `<BODY>`, `<SELECT>`;
`page-break-before:` and `page-break-after:` have values `auto`, `always`, `left`, or `right`, affect page breaks for document printing, apply to block-level elements and are not inherited;
`cursor:` defines the appearance of the cursor as it passes over the element where `cursor` was defined. It

applies to all elements, is inherited, and values are `auto`, `default`, `hand`, `crosshair`, `move`, `help`, `text`, `wait`, `e-resize`, `ne-resize`, `e-resize`, `se-resize`, `s-resize`, `sw-resize`, `w-resize`, `nw-resize` (the latter are compass directions).

There are also some pseudo-classes. `first-letter` and `first-line` can have all kinds of specifications (font etc.) and can be applied to all elements, for instance to paragraphs, even with a class:

```
P.initial:first-letter {font-size:3em;
     float:left}
```

will apply if we use `<P CLASS=initial>`.
Values are not inherited.

The tag `<A>` has four pseudo-classes, namely `link`, `active`, `visited`, `hover` (what happens when somebody points at the link). For instance:

```
A:hover {color:red; text-decoration:none}
```

Values are inherited, again, can be applied e.g. to `A.mine:active`.

6. Tricks

Your page contains a riddle and an answer, which is invisible until clicked on. How to do it? Write the answer as a link and set the background and links to the same colour, while visited link is different. Example:

```
<BODY BGCOLOR="#0000FF" LINK="#0000FF" ALINK="#FFFF00" VLINK="#FFFF00">
<A NAME="here"></A><A HREF="#here">Visible only after clicked on.</A>
```

Some nice applications of CSS: If we want background colour to extend somewhat beyond the text, we can define

```
.highlight {background-color:  yellow;
color:  black;
width:  10em;
border:  1px yellow}
```

and then use `<P><SPAN CLASS="highlight">`Text here`</SPAN></P>`.

Or we can have two different sets of link colours. Say:

```
A.set1:link {color:  red; background:  white}
A.set2:link {color:  blue; background:  white}
```

and then use `<A CLASS=set1 HREF= ... >`.

This is how to make paragraphs in two columns:

```
<DIV STYLE="float:  left; width:  50%">Paragraph 1</DIV>
<DIV STYLE="float:  right; width:  50%">Paragraph 2</DIV>
```

The following is not really a trick. The `<IMG>` loads many formats of pictures and one possible is GIF89a. Unlike a GIF87a format, the new one allows you to make simple animations. Basically, you create one picture which in fact is a series of several pictures that change in pre-set intervals. We thus get a so-called GIF-animation. The advantage is that a user's browser loads it as a picture just once and the rest is done by his viewer, so the animation continues to work without further downloads.

GIF-animations are done rather easily. First you prepare GIF-pictures needed for animation and then you combine them into one GIF using a GIF animation program. Each GIF file starts with a header: format (GIF97a or GIF89a), dimensions, color palette. After that comes the image. However, the header for animated GIFs has more info: each picture is preceded by a control block which (among other things) specifies the time delay. After the last image there is the GIF trailer and it is done.

The `<EMBED>` tag is supported by most major browsers but its definition is being sorted as of now. It is used to put objects (sound, video) directly into an HTML page. Some useful attributes:

— `SRC`
— `HIDDEN` with values `true` or `false`, can be used to hide for instance sound controls for background music.
— `AUTOSTART` with values `true` or `false`
— `LOOP` with values `true` (object repeats infinitely) or a natural number
— `WIDTH` and `HEIGHT` specify dimensions in pixels.

Of course, there is more to HTML than this. The paired `<APPLET>` tag includes a Java applet in the page— but there is no point in going into this without covering Java. When HTML, CSS and Java are put together we get what is called DHTML. The Dynamic HTML describes situation when we make pages that can change without contacting its server. As a matter of fact, the `A:hover` style definition is an example of DHTML, as another example we show the code for a rollover image in Appendix 1a. DHTML implementation is not standard right now, generally the more involved tricks one uses when creating apage, the higher the chance that on some browsers it will not show up right. And let it be a lesson to you.

Appendix 1: Form Fields

They serve as means of getting feedback. Such a field is delimited by <FORM>—</FORM>. Inside we create some dialogs using the following tags:

• The <INPUT> tag doesn't have an ending tag, the name of this field is specified by the attribute NAME. It also has an optional attribute TYPE. Its default value is
— text, which allows an input of a single line of text. The visible size can be defined using the SIZE attribute, longer inputted text should scroll horizontally. Maximal length (in number of characters) can be set up with the MAXLENGTH attribute and VALUE specifies text that should appear in the document when the document is first loaded.
Example: <INPUT TYPE=text SIZE=40 NAME=user VALUE="your name">
Other possible values are:
— password is just like text, but echoes typed characters with *.
— checkbox makes a checkbox. Its name is given by NAME and if a user checkes it, it is assigned the contents of the VALUE attribute. If the CHECKED attribute is used (a flag without any value), the checkbox is initialized in checked state. Multiple-choice boxes can be done by using several checkbox fields with the same NAME but different VALUEs, however, each of them generates a separate name/value pair in the submitted data.
— radio generates a single *name=value* pair from a set of alternatives. Each alternative is specified as a radio button with the same NAME and an explicit VALUE attribute. Unlike the multi-valued checkbox, here only the checked button in a group generates a name/value pair in the submitted data. At most one button in each group can have the CHECKED attribute flag. Example:

```
<INPUT TYPE=radio NAME=age VALUE="0-18">
<INPUT TYPE=radio NAME=age VALUE="19-35" CHECKED>
<INPUT TYPE=radio NAME=age VALUE="36-">
```

— submit defines a button that users can click on to submit the form's contents to the server. The button's label is set in the VALUE attribute. If the NAME attribute is used, the corresponding name/value pair will be included in the submitted data.
— image is like the submit thing, but instead of a clickable text string, the button is rendered as an image specified with the SRC attribute. The image can be aligned using ALIGN just like embedded images (see <IMG>), it also has the same attributes BORDER, HEIGHT, WIDTH, and HSPACE. In the submitted data, two name/value pairs are included, one for the x- and one for the y-coordinate of the clicked location. The names in this pair are created from the NAME attribute by appending .x and .y.
— reset creates a button that enables the user to reset the form fields to their initial state and does not create any output in the submitted data. Label can be specified using the VALUE attribute.
— file provides means of attaching files to the form's contents. Usually offers a text field (where the user can write the filename) with attributes as in the text type, and a button invoking a file browser. Optional attribute is ACCEPT, which contains a comma-separated list of MIME file types that are permissible.
— hidden is not rendered, so the user is not aware of it. Is is used to provide some info to a server without displaying it by the browser (or one can use a cookie to do this job).
— button just puts on a clickable button with the value of VALUE printed on it. Does nothing, but can be used with some Java Script.

• The <SELECT> tag requires </SELECT> and allows one to choose one of many or many of many options. It has the usual NAME attribute and one can specify the number of visible choices by the SIZE attribute. If not specified, the browser shows one choice with a "drop-down" list. Concerning how many items may be chosen, the default is "one of many", for "many of many" the MULTIPLE attribute flag must be used.
Options are specified between <SELECT> and </SELECT> using the <OPTION> tag. Each <OPTION> has the VALUE attribute which is submitted in case this option is selected. The text between <OPTION> and is shown to describe the option. An option can defined to be be marked as selected when the form is first loaded by using the SELECTED flag attribute.

• The <TEXTAREA> tag (with NAME attribute) offers a rectangular space where a user can type. It requires </TEXTAREA> and the text between these two is shown in the typing space when the document is first loaded. User can enter only text, the size of typing area is specified in <TEXTAREA> by attributes ROWS and COLS. The last attribute is WRAP, values are off (no wrapping), soft (or virtual), meaning the text is wrapped on the browser but sent as typed, and hard (or physical), meaning the text is sent not only with line breaks as typed, but also those caused by browser's line breaking.

Note that these tags are text-level tags, so we have to supply line-breaking and other things (like putting <TEXTAREA> into a separate paragraph). The action performed by the <FORM> tag is specified by ACTION and METHOD attributes. The type of HTML method is specified by METHOD, the default is get, when the server retrieves the information, possibly based on a query. The post alternative means that the info is stored somewhere and may elicit a response.

There are essentially four basic ways to process a form:
— Send the result via e-mail, say, `ACTION="mailto:foo@bar.com?SUBJECT=feedback"`. The contents of the form can be encrypted, the mechanism is specified by the attribute `ENCTYPE`. One popular choice is the following: `ENCTYPE="text/plain"` causes the form output to be arranged so that it can be read easier.
— Submit the form to a program stored on a server, traditionally a CGI script (Common Gateway Interface). In most cases, the program would process the form and send the results somewhere by e-mail. Example: `<FORM METHOD=post ACTION="http://www.acme.com/program.pl">`. Often the program is written in C or Pearl. Disadvantage is that this takes server's time and may overload it.
— Process the form with JavaScript. This is done by the browser, so it is nicer. Example: `<FORM METHOD="post" onSubmit="return form_check()">`
— Send the form to a database.

Example:
```
<H3>Example Reply Form</H3>
<FORM ACTION="mailto:user@provider.com" METHOD="post" ENCTYPE="text/plain">
<P><INPUT TYPE="hidden" NAME="subject" VALUE="Example Form">
Your age: <SELECT NAME="age">
<OPTION VALUE="under 18">under 18</OPTION>
<OPTION SELECTED VALUE="18 to 25">18 to 25</OPTION>
<OPTION VALUE="25 to 30">25 to 30</OPTION>
<OPTION VALUE="30 to 40">30 to 40</OPTION>
<OPTION VALUE="over 40">over 40</OPTION>
</SELECT>
<BR> Your name: <INPUT TYPE="text" NAME="name" size=40>
<BR> Your e-mai: <INPUT TYPE="text" NAME="email" size=40>
</P><P>
<TEXTAREA NAME="details" COLS=50 ROWS=10 WRAP="physical">More info</TEXTAREA></P>
<P>
I think this example is<BR>
<INPUT TYPE="radio" NAME="opinion" VALUE="great" CHECKED> great <BR>
<INPUT TYPE="radio" NAME="opinion" VALUE="middling"> middling <BR>
<INPUT TYPE="radio" NAME="opinion" VALUE="very poor"> very poor</P>
<P><INPUT TYPE="submit" VALUE="Process Data">  
<INPUT TYPE="reset" VALUE="Clear Data"></P>
</FORM>
```

When somebody uses this form, an e-mail message is generated which may look like this:
```
subject=Example Form
age=over 40
name=John Doe
email=jdou@DOA.com
details=This+is+a+test+and+only+a+test.
opinion=great
```